3

NOTE DI SOFTWARE

ARCHIVIO STORICO
ASSOCIAZIONE POZZO DI MIELE
FPDM-109
RNSW-101

UNIVERSITA'
DEGLI STUDI DI MILANO
ISTITUTO DI CIBERNETICA

Honeywell
Honeywell Information Systems Italia

Note di software.
è un bollettino trimestrale, edito a cura del Centro di Ricerca e Progettazione della Honeywell Information Systems Italia e dell'Istituto di Cibernetica dell'Università di Milano.

Note di software.
intende costituire uno strumento per la rapida e informale circolazione di informazioni, idee ed esperienze nell'area della tecnologia del software. In particolare, intende stimolare il dibattito sulle metodologie orientate alla diminuzione del rapporto costo/qualità dei programmi, e sugli strumenti atti ad automatizzare l'attività di produzione del software in ogni suo aspetto.

La collaborazione a **Note di software** è libera.
In particolare, si sollecitano contributi nella forma di brevi monografie o bibliografie essenziali commentate sui seguenti temi: tecniche e strumenti per la programmazione strutturata e modulare, metodologie e strumenti per il testing e il debugging dei programmi, documentazione del software, aspetti organizzativi e gestionali nella costruzione di sistemi software di grandi dimensioni.

Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Comitato di Redazione (HISI - Via ai Laboratori Olivetti - Pregnana Milanese), o di inviare direttamente il dattiloscritto in triplice copia.
La preparazione del testo nel formato adatto alla fotoriproduzione è a carico degli autori, ai quali verranno comunicate le norme redazionali.
I contributi non dovrebbero superare le quattromila parole.

I lavori accettati per la pubblicazione non vengono revisionati da un comitato tecnico. Pertanto, ogni contributo pubblicato riflette le opinioni personali dei suoi autori, e non necessariamente quelle del Comitato di Redazione.
Eventuali revisioni saranno effettuate di comune accordo fra il Comitato di Redazione e i proponenti.

Note di software.
viene distribuito a chi ne faccia richiesta alla redazione.

Direttore responsabile: Paolo Ghelfi
Comitato di Redazione: Roberto Polillo - Wanda Anselmetti
Redazione: Franco Soresini

3

FPDM-109

Luglio - Settembre 1977

NOTESOFTWARE

Indice:

QUESTO NUMERO . . . . pag. 1

DISCUSSIONI:

- DOCUMENTAZIONE E PRODUTTIVITA', di G. Degli Antoni, G. Torriani " 2

CONTRIBUTI TECNICI:

- LA REALIZZAZIONE DELLA DOCUMENTAZIONE PER UTENTI DEL LIVELLO 62, di G. Armanini, R. Candela, S. Leva, R. Mari, S. Mazzocchi " 10
- AUTODOCUMENTAZIONE DEI PROGRAMMI, di G. Gobbi, M. Maiocchi " 16
- DOCUMENTAZIONE PER L'UTENTE: ALCUNE INDICAZIONI E UN ESEMPIO, di A. Maserati, S. Mazzocchi, R. Polillo, G. Torriani " 22
- VALUTAZIONE E MIGLIORAMENTO DELLE PRESTAZIONI DI STNAL, di L. Becattini, F. Faidutti " 31
- UN'OPPORTUNA STRUTTURA DI DATI PUO' SEMPLIFICARE LE STRUTTURE DI CONTROLLO? , di L. Antonelli " 37
- TMSS: UNO STRUMENTO PER LA GESTIONE AUTOMATICA DI CHECK-LIST DINAMICHE NELLA QUALIFICAZIONE INDUSTRIALE DEL SOFTWARE, di A. Cazziol, C. Cucciati " 41

AVVENIMENTI " 52

IL SEGNALIBRO " 54

Autorizzazione del Tribunale di Milano n. 87 del 23 febbraio 1977

QUESTO NUMERO . . .

Con questo terzo numero, NOTE DI SOFTWARE si propone di iniziare un dibattito sui problemi e sulle tecniche della documentazione del software, nei suoi vari aspetti.

I lavori sulla documentazione che qui vengono presentati sono quattro. Il primo (Degli Antoni e Torriani) fornisce una introduzione ai lavori che seguono, inquadrando l'attività di documentazione nell'ambito complessivo del processo di produzione del software, ed esaminando i rapporti fra qualità della documentazione e produttività.
Seguono due lavori sulla documentazione per l'utente di un prodotto software. Il primo (Armanini, Candela, Leva, Mari e Mazzocchi) descrive gli aspetti organizzativi del processo di costruzione e manutenzione dei manuali per l'utente del Livello 62. Il secondo (Maserati, Mazzocchi, Polillo e Torriani) fornisce invece un insieme di indicazioni di metodo per la costruzione di manuali per l'utente che risultino di facile uso e possano essere utilizzati, senza sostanziali costi aggiuntivi, nell'attività di formazione sul prodotto. Queste indicazioni vengono illustrate facendo riferimento al manuale utente del Transactional Program System del Livello 62.
Gobbi e Maiocchi, infine, definiscono i requisiti di quella parte di documentazione tecnica volta a semplificare l'attività di manutenzione dei programmi, e propongono uno standard di "autodocumentazione", per cui il codice stesso del programma, arricchito di commenti opportunamente strutturati, costituisce il documento principale per il manutentore.

Gli altri tre contributi trattano altri aspetti dell'ingegneria del software. Cazziol e Cucciati, dopo una panoramica sui problemi di tipo gestionale e organizzativo posti dall'attività di qualificazione di un sistema software di grandi dimensioni e che evolve nel tempo per release successive, descrivono uno strumento particolare, TMSS, che permette di gestire in modo automatico check-list statiche e dinamiche.
STNAL, un insieme di macro che rendono disponibili le principali strutture di controllo di alto livello al programmatore assembler NAL, è l'argomento del lavoro di Becattini e Faidutti. In esso, vengono descritti i risultati di un'attività volta alla valutazione e al miglioramento delle prestazioni delle macro, per ridurne i tempi aggiuntivi di compilazione dovuti alla presenza di un macroprocessor.
La strutturazione di un programma nella fase di disegno è, infine, il tema del lavoro di Antonelli. Esso mostra, con un esempio, come un opportuno disegno delle strutture di dati di un programma possa, a volte, semplificarne notevolmente la struttura di controllo.

(La Redazione)

DISCUSSIONI

DOCUMENTAZIONE E PRODUTTIVITA'(+)

G.Degli Antoni (°) - G. Torriani (°°)

1. Introduzione

La moderna tecnologia ha abituato alla realizzazione di opere complesse che debbono la loro esistenza alla divisione del lavoro.
Grazie ad essa, compiti estremamente complessi vengono suddivisi in compiti di dimensione umana. Questo è ad esempio il caso di progetti spaziali, del progetto e realizzazione di sistemi di calcolo o di ospedali o di complessi industriali ed ancora della progettazione di automobili.

Ma tale divisione del lavoro non è priva di effetti negativi secondari, che ne limitano la portata ed inducono difficoltà di vario genere.

Se si escludono le conseguenze di una esasperata divisione del lavoro , che conduce alla definizione di ruoli al di sotto delle possibilità umane, una analisi delle difficoltà segnalate porta facilmente a concludere che queste sono spesso legate a insufficienze delle comunicazioni fra i gruppi che partecipano alla realizzazione dell'opera. Così, ad esempio, la mancanza di una visione complessiva da parte di questi impedisce l'espletamento di autocritica e di creatività, e favorisce d'altro canto l'insorgere di errori.

Il compito di ridurre tali difficoltà è generalmente devoluto alla documentazione e ad una vasta attività che utilizza la documentazione stessa come punto di riferimento.

Tale documentazione ha dunque un compito complesso,difficile e diversificato. A tale compito potrà assolvere solamente se sarà realizzata con la cura che questo richiede.

(+) Il presente lavoro è già apparso,in una precedente versione,negli atti del XIV Convegno Internazionale di Automazione e Strumentazione (FAST), Milano,23-24 novembre 1976

(°) Università di Milano - Istituto di Cibernetica
(°°) Honeywell Information Systems Italia

Nella convinzione che spesso la documentazione è trascurata, la presente nota ha lo scopo di sottolineare succintamente l'importanza che essa riveste nei processi che ne fanno uso e di fornire alcuni suggerimenti rivolti a migliorare vari aspetti dell'attività produttiva.

2. La documentazione

La documentazione nei processi produttivi viene utilizzata per comunicare ciò che il processo produttivo deve fornire, le modalità con cui ottenerlo, le modalità di impiego dei prodotti.
Tale letteratura è destinata a varie classi di lettori con differenti livelli di dettaglio. Questi dipenderanno dalla struttura (gerarchica) del processo produttivo.

Generalmente parlando, la documentazione sarà un'espressione in un opportuno linguaggio (quasi mai formalizzato), che rispecchi il rapporto fra mittente e destinatario. Così, ad esempio, la documentazione dell'ufficio progetti per la direzione aziendale rifletterà i risultati del primo e le aspettative della seconda, in un linguaggio non necessariamente tecnico e a meno di molti dettagli che finirebbero coll'offuscare il messaggio stesso della documentazione.

Accanto agli aspetti d'uso ora accennati, la documentazione presenta aspetti relativi alla modalità con cui viene apprestata.

Così, è chiaramente indispensabile disporre di documentazione su ciò e su come si intende produrre, prima di avviare completamente il processo produttivo. Ma è anche inevitabile che l'avviamento di questo introduca varianti che impongono modifiche alla documentazione. Si sviluppa pertanto un processo produzione-documentazione che alla fine fornirà prodotti e la corrispondente documentazione.
La relazione fra processi di produzione e documentazione potrà essere di mutuo accoppiamento o di semplice dipendenza. Quest'ultimo è in sostanza il caso in cui uno dei due processi non influenza l'altro. Qualora i due processi siano mutuamente accoppiati, la distinzione (arbitraria) fra documentazione e produzione introduce una retroazione capace di migliorare la qualità del prodotto e della documentazione.

Ciò sostanzialmente indica una via attraverso cui ottenere la correttezza dei prodotti: tale via è lo sviluppo concatenato dei due processi con una continua verifica della correttezza relativa (ovvero della corrispondenza fra ciò che deve essere prodotto e la sua documentazione).

Il mutuo accoppiamento fra produzione e documentazione non annulla tuttavia le difficoltà. Infatti la documentazione viene talvolta utilizzata allorchè non tutte le verifiche sono state effettuate, con inevitabili conseguenze nel caso di errori o di insufficienza di comunicazione.

3. Interazione con la documentazione

La documentazione di prodotti tecnologici tende a produrre nei suoi lettori reazioni non molto dissimili da quelle osservate negli utenti dei prodotti stessi.
Ad esempio, come è ben noto, spesso gli utenti di un prodotto si limitano a considerarne gli aspetti relativamente più importanti e ne trascurano molti altri, ritenuti secondari ma che comunque finiranno con l'avere influenza sull'uso del prodotto.
La stessa situazione si presenta frequente nel caso della documentazione. Il lettore di un manuale, in genere, ne considera con attenzione solo una parte limitata. Molti dettagli verranno del tutto trascurati nel contenuto e talvolta ne verrà ignorata anche la presenza.

Le ragioni di questa situazione sono molte e non facilmente eliminabili.
Da un lato, chi appresta la documentazione tende a fornire un risultato completo di tutti i dettagli del caso, anche quando questi sono la conseguenza diretta del senso comune, o comunque hanno carattere culturale e poco riflettono la creatività di chi ha contribuito al prodotto documentato.
Dall'altro, il lettore non tende a rinunciare ai propri mezzi creativi e si porrà nell'ottica di avere compreso la documentazione appena le letture effettuate gli permetteranno di individuare l'ambito culturale in cui collocare i concetti in esame.

Evidentemente, le ragioni di questa situazione sono legate alla variabilità della classe dei lettori. La miglior definizione di questi permetterebbe in linea di princi-

pio di ridurre le incertezze considerate, ma richiede di introdurre documentazione differenziata per le differenti classi di lettori.

Una via per ridurre le difficoltà segnalate è quella di incoraggiare il lettore a concettualizzare egli stesso il più possibile il mondo di cui la documentazione tratta, lasciandogli la possibilità di verificare i dettagli solo quando l'esperienza ne metta in evidenza l'importanza.
Ciò implica un'accurata analisi dei modi e dei tempi di esposizione nell'ambito dei soggetti trattati, dell'impiego di esempi, ecc., oltre ad un modo accurato ed efficiente per indirizzare il lettore verso i dettagli voluti.
La costruzione di guide per il lettore, semplici, chiare ed immediate, rappresenta un elemento della soluzione del problema.
In questo ordine di idee viene talvolta suggerita l'adozione dei testi programmati e della documentazione assistita da elaboratore.
Ma ancora una volta l'adozione di interazione fra chi realizza documentazione e prodotti può permettere di introdurre fra gli attributi di questi ultimi la nozione di "documentabilità".
Tale attributo è sostanzialmente la base del concetto di "end user" che viene utilizzato nel progetto di sistemi informativi i cui utenti dispongono di una competenza minima (del sistema).

4. Effetti collaterali della documentazione

Gli effetti della documentazione sono molteplici e vanno ben oltre gli scopi immediati per cui la documentazione viene prodotta.

Infatti lo sviluppo di sistemi sempre più complessi fa sì che la documentazione diventi la via attraverso cui vengono istruite persone non solo a fare cose, ma anche a conoscere.
Se poi si considera che spesso la documentazione tecnica è una frazione non piccola della letteratura che il personale tecnico finisce con il leggere, ne segue a maggior ragione tutta una classe di effetti secondari che non è negli scopi di queste note analizzare, ma solamente segnalare.

Ma certamente non è secondario osservare che le competenze del personale tendono a costituirsi attraverso la sola via della lettura della documentazione. E se ciò può essere relativamente produttivo in un certo periodo della vita di un'azienda, diventerà causa di obsolescenza allorchè nuove tecnologie imporranno nuovi prodotti la cui comprensione richiede nozioni di base che sono state ritenute inessenziali nella documentazione tecnica e che la scuola non ha fornito.
Il fatto che la documentazione diventi la base per l'apprendimento attraverso la esperienza deve quindi suggerire molta attenzione ai responsabili della sua realizzazione, che non potranno trascurare aspetti strategici a proposito del futuro aziendale.

Ma l'importanza della documentazione non è tutta legata alla formazione dell'esperienza dei tecnici.
Infatti, ad esempio, gli utenti dei prodotti potranno identificare un'immagine dei prodotti ed anche aspetti della società che li ha forniti attraverso la lettura della documentazione.
L'immagine dei prodotti e dell'azienda non può non influire su aspetti di diffusione degli stessi, nel caso in cui questa non sia limitata da altri fattori (la qualità,ecc.).

5. Documentazione e formazione

Oltre che attraverso l'uso, la documentazione tecnica determina competenza come supporto di attività di formazione, sia di personale tecnico che di utenti di sistemi.
In tutti i casi, la documentazione avrà caratteristiche legate al tipo di impiego e differirà solamente per livelli di dettaglio o stile espositivo. Inoltre, sarà integrata da altri documenti , quali i manuali per l'istruttore o per i candidati.

E' difficile affermare che tale situazione può essere convenientemente mutata, ossia che è possibile introdurre manuali validi sia come manuali formali per la documentazione di prodotti o di modi d'uso, sia come testi per l'istruzione. Tuttavia è opportuno segnalare esplicitamente che, qualora lo sforzo venga tentato, ne risulterebbero alcune conseguenze positive:

- Lo sviluppo dei testi di istruzione può accompagnare lo sviluppo di prodotti, realizzando ancora una volta processi mutuamente accoppiati, con beneficio sulla reciproca correttezza.
- I manuali perdono quel carattere "manualistico" che talvolta li rende poco accettabili ai lettori.
- Risulta possibile ai progettisti di documentazione effettuare attività didattica.

L'importanza della documentazione sulla formazione di personale è anche legata al modo di produrla. Ciò è vero soprattutto per la documentazione tecnica, il cui sviluppo può diventare una delle vie per la formazione di personale da dedicare alla progettazione.
Ad esempio è possibile dedicare personale non particolarmente dotato di esperienza alla verifica della correttezza di prodotti attraverso il continuo riscontro con la documentazione, che sarà apprestata da personale sulla cui esperienza invece non si insisterà mai a sufficienza.
In questo modo il personale non particolarmente dotato di esperienza può venir inserito nel progetto in modo efficace con ripercussioni immediate sulla qualità dei prodotti oltre che sulla propria diretta formazione. Naturalmente la sua esperienza crescerà con lo sviluppo del progetto ed alla fine quel personale sarà disponibile sia per attività di progettazione o di realizzazione che per attività di documentazione e formazione.
Questo modo di introdurre nuovo personale nel processo produttivo non annulla la necessità di sforzi di formazione di base iniziali.

6. Documentazione e partecipazione

La partecipazione al processo produttivo è una delle vie di accesso alla qualità del lavoro. Tale partecipazione è la base per la creatività, dalla quale può dipendere, ad ogni livello, il successo del processo produttivo.

D'altra parte, la partecipazione può essere ottenuta solamente attraverso una diffusa conoscenza del progetto che ne fornisca una chiara immagine complessiva.

Il compito di fornire tale immagine è ancora da assegnare alla documentazione o ad una documentazione specifica. Tale documentazione ha un compito arduo: la descrizione di un sistema in fase di sviluppo, a meno dei dettagli, e a un livello di astrazione, seppur altro, sufficientemente accurato da rappresentare un'interfaccia logica per tutti i partecipanti al progetto, cosicchè questi sappiano collocare con accuratezza la propria attività. E il tutto a un costo moderato.

Una simile documentazione è difficile da ottenere, e pressochè impossibile in tutti quei casi in cui cause di forza maggiore (errori, cambiamenti di indirizzo,ecc.) determinano un'evoluzione non prevista nello sviluppo del progetto.
In questi casi, lo sviluppo del progetto tende a trasformarsi in sviluppo bottom-up anche se era stato impostato come sviluppo dop-down. La relativa documentazione non può subire una sorte differente. Anzi, talvolta l'emozione determinata da possibili rischi di ritardo farà sì che la struttura della documentazione rifletterà con qualche ritardo le scelte, conseguenti all'evento, che hanno introdotto una svolta evolutiva del prodotto.

Inoltre, la partecipazione al progetto diventa più difficile a causa delle insufficienze di comunicazione sullo stato dello stesso.

7. Conclusione

La qualità della documentazione è uno degli elementi essenziali della produttività dei processi che ne fanno uso.

La produttività infatti è legata al processo decisionale che determina le varie scelte. Naturalmente non tutte le scelte dipenderanno da ingredienti comunicati esplicitamente attraverso la documentazione.
Talvolta tali ingredienti (dati, nozioni, affermazioni, procedimenti, ecc.) saranno stati acquisiti attraverso letture precedenti o senza l'ausilio della documentazione.
Ne segue che anche tenuto conto di queste situazioni, la produttività del processo decisionale è legata in modo diretto alla qualità della documentazione. In tutti quei casi in cui il processo decisionale è una frazione non trascurabile del processo complessivo, la documentazione finirà con il determinare la produttività attraverso la qualità.

Le presenti note hanno anche discusso altre influenze della qualità della documentazione o del suo modo di produrla sul processo produttivo. Volendo includere nella produttività tutti gli effetti (benefici) legati alla qualità della documentazione, si può concludere che la produttività in genere aumenta. Infatti, con la qualità della documentazione :

. migliora la possibilità di inserimento di nuovo personale nel processo produttivo
. migliora la competenza del personale e quindi la produttività a causa della migliorata capacità produttiva
. migliora la produttività del processo decisionale
. lo sforzo rivolto alla formazione di personale è destinato a ridursi
. la diffusione dei prodotti tende a migliorare
. vengono ridotti i costi di integrazione
. la qualità del lavoro tende a migliorare con positive influenze sulla quantità di lavoro effettuato per unità di tempo
. le modifiche risultano meno drammatiche migliorando la comunicazione fra i partecipanti al progetto.

Si noti che abbiamo del tutto trascurato l'analisi dei problemi di manutenzione. Ciò è la conseguenza di aver considerato tale attività come un tipico uso dei prodotti, e la corrispondente documentazione fra i documenti d'uso degli stessi, con effetti quindi compresi nella discussione precedente.
Una menzione merita infine il rapporto fra qualità della documentazione ed evoluzione del prodotto.
E' difficile affermare quali siano le conseguenze della qualità della documentazione sull'evoluzione del prodotto. Si possono infatti immaginare due tendenze contrastanti: la qualità della documentazione da un lato agevola per tutti la scoperta delle inadeguatezze del prodotto rispetto a future esigenze; dall'altro ne facilita il superamento mettendo in evidenza le informazioni necessarie.
A seconda che prevalga la prima o la seconda delle due tendenze si determineranno condizioni tendenti a proporre o meno una evoluzione.

Va da sè che l'analisi proposta tende a sottolineare il ruolo della documentazione nella formazione di opinioni sulle future linee di sviluppo di un prodotto. Naturalmente il successo di tale ruolo dipenderà dalle complesse situazioni in cui il prodotto si trova a vivere.

CONTRIBUTI TECNICI

LA REALIZZAZIONE DELLA DOCUMENTAZIONE PER UTENTI DEL LIVELLO 62

G.Armanini (°+),R.Candela(°+),S.Leva(°),R.Mari(°+),S.Mazzocchi(°+)

Riassunto. La documentazione rivolta all'utente di sistemi di calcolo e' parte integrante dell'attivita' complessiva necessaria alla produzione di un calcolatore. Questo articolo descrive come il problema della documentazione e' stato affrontato nell'ambito del Livello 62. Viene anche fornita una descrizione della procedura e degli strumenti per produrre i manuali d'uso del sistema.

1. ASPETTI DELLA DOCUMENTAZIONE USER

L'esperienza, di cui tratta la presente nota, si riferisce alla realizzazione di documentazione per utenti di sistemi di calcolo della Serie 60 Livello 62 progettati a Pregnana Milanese. In quel laboratorio, l'attivita' di documentazione e' una delle numerose attivita' associate allo sviluppo del software e dell'hardware che procede per fasi successive. Cio' dipende dal fatto che il sistema nel suo complesso evolve nel tempo e che vengono, per esso, pianificate delle scadenze di distribuzione al pubblico degli utilizzatori.

Lo scopo di questo lavoro e' quello di abbozzare un'analisi e di descrivere uno dei processi che sempre si accompagnano alla produzione di sistemi di calcolo: la documentazione per l'utente.

La documentazione user e' costituita da quell'insieme di informazioni che permettono di fornire all'utente, assieme ad una descrizione generale di tutte le funzionalita' del sistema, le norme d'uso di ogni singolo prodotto.

Un problema che caratterizza il processo di realizzazione della documentazione e' il fatto che essa continua ad evolvere nel tempo proprio perche' e' suo compito quello di riflettere nel tempo lo stato del software che essa descrive.

Per l'ambiente che si vuole descrivere, cioe' il Livello 62, la documentazione user e' composta, ad oggi, di 24 manuali, per un totale di circa 5000 pagine. I manuali, tranne qualcuno che introduce i concetti generali del sistema, descrivono ciascuno funzionalita' distinte del software.

Risulta da cio' evidente che esiste una stretta relazione tra sviluppo del prodotto software e la sua documentazione. In particolare e' necessario che qualunque aggiornamento ai prodotti esistenti od un nuovo prodotto sia documentato. Cio' comporta due tipi di attivita':

- l'aggiornamento dei manuali esistenti
- la stesura dei manuali nuovi.

Il processo di produzione della documentazione user, per quello che si e' detto, coinvolge numerose entita' che concorrono alla realizzazione del software del calcolatore.

2. ENTITA' CHE CONCORRONO ALLA DOCUMENTAZIONE

Un'analisi dei vari gruppi di lavoro puo' essere interessante perche' e' proprio la gestione di processi molto articolati che sovente crea problemi organizzativi.

Le entita' in gioco nella realizzazione della documentazione user sono principalmente:

(°)H.I.S.I.- Pregnana Milanese

(+) Istituto di Cibernetica- Milano

progettisti o sviluppatori del sistema, i quali forniscono la quasi totalita' delle informazioni sulle specifiche del prodotto e sulle norme operative. Nell'ambiente descritto, essi sono solitamente uniti in gruppi di lavoro a ciascuno dei quali e' affidata lo sviluppo o la manutenzione di un prodotto

l'organizzazione marketing la quale assolve principalmente due compiti: quello di indicare le necessita' dell'utenzm (ad esempio suggerendo di migliorare o completare certi manuali) e quello di indicare quale visibilita' del prodotto deve essere data all'utente. Infatti l'organizzazione marketing e' l'organismo direttamente a contatto con l'utente, ne conosce quindi i problemi e le necessita', ed e' in grado di decidere su quali aspetti del sistema concentrare l'attenzione dell'utente e quali aspetti l'utente puo' ignorare. Inoltre, l'organizzazione marketing cura i rapporti con le corrispondenti organizzazioni all'estero, e si fa portavoce delle loro esigenze.

un gruppo di technical authors il cui compito e' principalmente quello di rielaborare le informazioni provenienti dagli sviluppatori del prodotto, in accordo con gli orientamenti espressi dal marketing, e di curare l'edizione delle pagine dei manuali.

In effetti lo sforzo dei technical authors e' piu' articolato in quanto deve raggiungere vari obiettivi:

- trasformare le specifiche funzionali (che sono documenti tecnici ad uso esclusivamente interno) in documenti comprensibili all'utente

- assicurare la completezza e l'organicita' delle informazioni descritte in modo da ridurre gli interventi di supporto all'utente da parte del marketing

- assicurare la distribuzione omogenea delle informazioni su una stessa funzionalita' di sistema in tutto il set della documentazione, per evitare che informazioni riguardanti prodotti che vengono descritti in piu' di un manuale siano in conflitto

- garantire che tutte le variazioni del prodotto che intercorrono tra le specifiche funzionali originarie e il prodotto effettivamente distribuito siano documentate.

un gruppo di clerical writers che hanno il compito di curare l' "editing" automatico dei manuali (cfr. paragrafo 4.) e che interfacciano esclusivamente con i technical authors.

Le entita' che concorrono alla documentazione sono numerose, come abbiamo visto. Per cui, evidentemente, il flusso di dati che queste si scambiano e' complesso e molto articolato. Il data-flow di figura 1 pero' chiarisce, anche senza un ulteriore commento, che tipo di dati vengono scambiati tra le varie entita' e in quale direzione avviene questo scambio.

3. ORGANIZZAZIONE DELL'ATTIVITA'

Ne risulta che al processo di documentazione concorrono numerose entita' e che e' percio' necessario un corretto coordinamento delle divese attivita'.

Per questo, il lavoro di documentazione ha richiesto:

strumenti di lavoro efficaci e consolidati dall'esperienza.

una procedura estremamente rigorosa anche se flessibile e dinamica

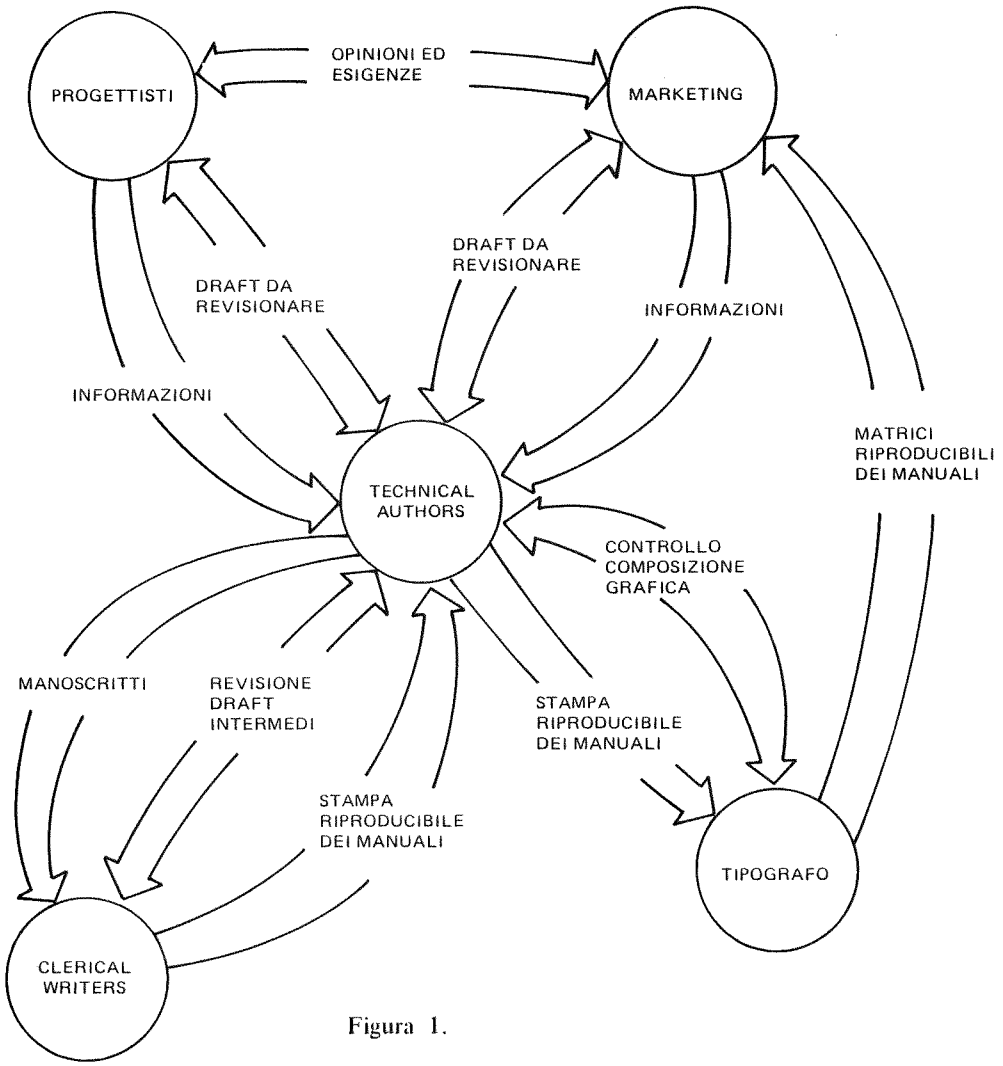


Figura 1.

Strumenti per la raccolta delle informazioni.

Questi sono:

un documento contenente le specifiche funzionali delle variazioni del software rispetto alla precedente release del sistema (system release specifications) emesso dai responsabili del servizio ingegneria

le specifiche funzionali di ciascun componente del sistema (functional specifications) emesse dai gruppi responsabili di ogni prodotto

per i manuali nuovi, una out-line emessa dal marketing che descrive i contenuti generali e l'impostazione che i manuali avranno

questionario inviato ai project leaders su cui essi e gli sviluppatori danno una stima, in termini di argomenti e di pagine, della quantita' di aggiornamenti che prevedono per la prossima release, e nello stesso tempo indicano la data entro cui i technical authors riceveranno tali informazioni

bozze (draft) successive continuamente revisionate da technical authors, MKTG e sviluppatori fino alla stesura del draft definitivo.

Piani di lavoro. In base alla valutazione delle informazioni raccolte, in base alla data di scadenza della release e alle priorita' di emissione dei manuali, i technical authors emettono un piano di lavoro che prevede, settimana per

settimana, per ogni manuale, il tipo di attivita' che bisogna svolgere su di esso.

Queste attivita' possono essere riassunte nei seguenti punti:

- stesura iniziale
- memorizzazione del testo su memoria di massa (vedi descrizione del Text-Editor piu' avanti)
- revisione da parte degli sviluppatori
- revisione da parte del marketing
- revisione da parte delle consociate
- aggiornamento del testo in memoria
- stampa finale del manuale
- apposizione di simboli speciali, disegni etc. e produzione di matrici riprodkcibili, da parte di una tipografia.

4. STRUMENTI PER L'AGGIORNAMENTO, MEMORIZZAZIONE E STAMPA DEI TESTI

Gli strumenti usati per memorizzare e stampare i manuali user sono un elaboratore Honeywell 66 e un insieme di terminali remoti Terminet 300 (od altri terminali aventi caratteristiche analoghe) ad esso collegati in Time-Sharing.

Il sistema software utilizzato e' il Text-Editor che e' un sottosistema del Time Sharing costituito da due programmi:

- EDITOR che permette di memorizzare, correggere ed aggiornare i flussi permanenti su cui si memorizzano le pagine dei manuali
- RUNOFF che permette di stampare i flussi stessi, in un formato scelto. Cio' si ottiene interpretando alcune parole di controllo, inserite precedentemente nel testo, che dirigono la formattazione e la stampa.

Il presente articolo e' un esempio d'uso del Text-Editor: esso e' infatti stato memorizzato, aggiornato in memoria in fasi successive di revisione, e stampato tramite un Terminet 300 collegato ad un Honeywell 66.

Il testo di un manuale puo' essere memorizzato in uno dei seguenti modi:

- tramite l'immissione dalla tastiera di un Terminet 300
- tramite l'immissione da paper tape precedentemente preparato mediante il medesimo terminale.

La figura 2 sintetizza il processo precedentemente visto.

5. PROCEDURA

La procedura di lavoro e' schematizzata nel flow-chart di figura 3.

6. MANIPOLAZIONE DELLE INFORMAZIONI MEMORIZZATE

Attraverso il terminale il testo puo' essere richiamato in memoria ed aggiornato per mezzo di un insieme di semplici comandi dell'Editor. Questo lavoro e' svolto da clerical writers che ricevono dai technical authors gli aggiornamenti da apportare ad un dato manuale.

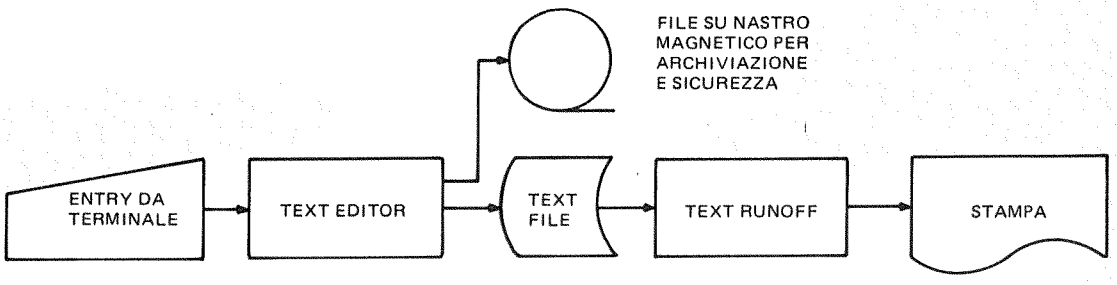
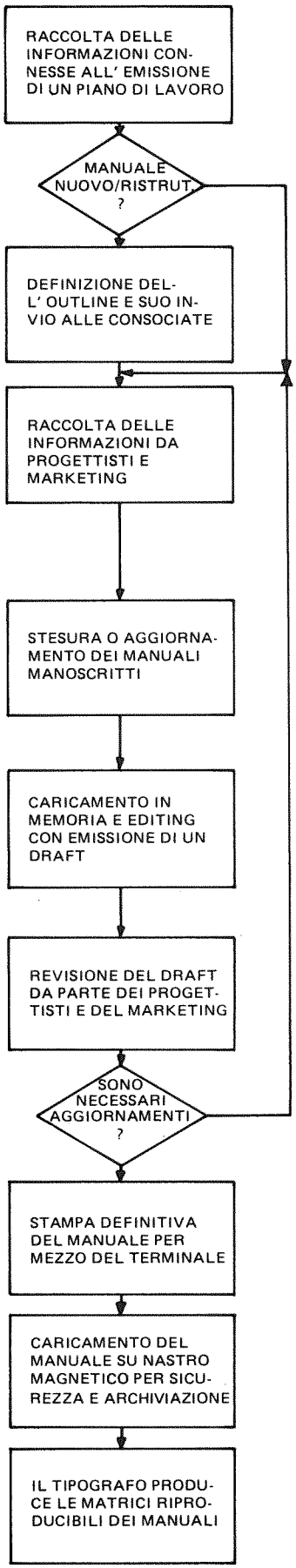


Figura 2.



Questo passo preliminare e' necessario per l'organizzazione di tutto il lavoro successivo. Tutte le entita' sono coinvolte.

Se un manuale e' nuovo o e' da ristrutturare, viene preparata una out-line da inviare alle consociate per eventuali commenti.

E' il nucleo centrale del lavoro: i technical authors, progettisti e marketing hanno una fitta rete di incontri per scambiarsi informazioni tecniche e opinioni.

I technical authors elaborano le informazioni fornendo dei manoscritti ai clerical writers.

I clerical writers caricano in memoria i manoscritti e forniscono ai technical authors dei draft.

Un altro step fondamentale della procedura: progettisti e marketing revisionano i draft fornendo informazioni e pareri.

Se progettisti e marketing hanno dato il benestare definitivo, il manuale viene stampato in copia unica riproducibile,

caricato su nastro e

mandato al tipografo perche' produca delle matrici.

Figura 3.

I clerical writers producono, seguendo degli standards editoriali, un draft successivamente rielaborato dai technical authors. Questa operazione viene ripetuta piu' volte fino all'ottenimanto del draft finale.

Quando, seguendo le scadenze del piano di lavoro, un manuale ha ricevuto il benestare dei progettisti e del supporto tecnico del marketing, i clerical writers producono, a terminale, una stampa definiva del testo.

Nello stesso tempo, il manuale e' memorizzato su nastro magnetico per motivi di sicurezza e di archiviazione.

La stampa definitiva passera' allo studio tipografico che appone simboli speciali, esegue disegni e tabelle; da queste pagine composte verrannno quindi prodotte delle matrici distribuite a tutte le consociate.

7. STANDARDS DI DOCUMENTAZIONE

I technical authors ed i clerical writers seguono, per le parti di loro competenza, dei precisi standards di documentazione. Questi standards sono stabiliti e seguiti per la documentazione di tutti i Livelli della Serie 60.

Le indicazioni fornite dagli standards sono duplici e riguardano:

aspetti di contenuto:

- gli standards prevedono che il sistema sia descritto per argomenti a ciascuno dei quali e' dedicato un manuale; esiste, cioe', un disegno generale di come ripartire i vari aspetti del sistema all'interno del set dei manuali
- per ogni manuale viene indicata una precisa struttura interna, vale a dire una ripartizione degli argomenti in sezioni ed in appendici

aspetti editoriali: gli standards indicano in modo esatto la veste tipografica delle pagine, con un dettaglio che investe dimensioni delle parti scritte, delle figure e caratteristiche di eventuali simboli speciali (ad esempio i simboli per la stesura dei flow-chart).

8. CONCLUSIONI

In questo articolo e' stata presentata la procedura di lavoro con cui i Technical Authors curano l'edizione dei manuali del Livello 62. Non ci si e' voluto soffermare a lungo sugli aspetti metodologici e contenutistici della documentazione per l'utente poiche' si e' voluto enfatizzare sopratutto le tecnologie e gli strumenti usati in quel lavoro. In particolare il Text-Editor e' stata una scelta opportuna poiche' esso permette non solo di memorizzare una quantita' di testi e documenti, ma anche di aggiornarli continuamente ed avere immediata stampa di essi in qualunque momento.

AUTODOCUMENTAZIONE DEI PROGRAMMI

G. Gobbi (°) - M.Maiocchi (+)

Riassunto

Si esaminano alcuni aspetti della documentazione dei programmi con particolare riferimento ai problemi di manutenzione.

Si propone un metodo di autodocumentazione che si è rivelato utile a migliorare la mobilità dei programmatori sui progetti, a favorire la crescita di competenza del personale, a ridurre i costi di modifiche o correzioni e i costi di documentazione.

1. Introduzione

In ambienti di produzione di software di dimensioni medie o grandi la documentazione dei prodotti riveste un ruolo di primaria importanza non solo ai fini di corredare il prodotto della necessaria descrizione, ma anche, fondamentalmente, in quanto essa è il principale strumento di comunicazione tecnica che permette di diffondere competenza e cultura e che permette di integrare i progettisti nell'ambiente fisico di lavoro.

In questo senso soprattutto possono essere esaminati i documenti di "specifiche funzionali", che descrivono le caratteristiche funzionali che un prodotto offre (o dovrà offrire), le specifiche di disegno, che ne descrivono l'architettura generale e la collocazione nell'ambito del sistema, e le "specifiche" che potremmo definire di "maintenance", volte a descrivere la struttura interna del prodotto, con un dettaglio sufficiente a permettere interventi sul prodotto anche da parte di chi non abbia partecipato alla sua costruzione.

(°) Honeywell Information Systems Italia - Pregnana Milanese

(+) Istituto di Cibernetica - Università degli Studi di Milano e Honeywell Information Systems Italia - Pregnana Milanese.

Il presente lavoro si occupa proprio delle specifiche volte alla maintenance, esaminandone gli obiettivi, i problemi che la loro stesura in modo tradizionale impone, ed infine proponendo una soluzione alle difficoltà che avremo esaminate.

2. Scopi della documentazione volta alla maintenance.

Obiettivo primario dichiarato della documentazione di maintenance è quello di permettere una adeguata manutenzione dei prodotti.

Attualmente si riconosce il costo di manutenzione come il maggiore durante tutta la vita di un programma [5]: tale costo è determinato dal fatto che gli interventi in un programma ormai stabile sono un fatto che accompagna il programma per tutta la sua vita, sia a causa di richieste di cambiamenti funzionali dovuti a cambiamenti dell'ambiente circostante (sia software che hardware), sia a causa del reperimento di errori durante il suo uso da parte dell'utente esterno. In entrambi i casi gli interventi si collocano in genere temporalmente molto distanti dal momento del disegno e dell'implementazione dei programmi, così da creare problemi nella loro conoscenza di dettaglio, anche per aspetti di cambiamento di personale. Ciò è tanto più costoso quanto meno organizzati e strutturati si presentano i programmi: una porzione di codice riutilizzata da più parti di un programma mediante l'uso di variabili switches rischia di portare a correzioni che introducono nuovi errori, e le situazioni analoghe a questa sono molto frequenti.

E' fondamentale quindi una buona documentazione che guidi un qualunque progettista alla comprensione di un prodotto che egli non ha implementato.

Un altro aspetto rilevante per determinare l'esigenza di buona documentazione per la manutenzione è quello che potremmo definire "portabilità" dei programmatori sui vari progetti.
In una azienda dove l'implementazione sia distribuita tra numerosi progettisti è importante poter rapidamente sostituire personale venuto a mancare su un progetto, senza che tale introduzione causi un rallentamento eccessivo per problemi di "messa al passo" del nuovo arrivato; inoltre un'adeguata mobilità del personale sui vari progetti permette una miglior distribuzione delle forze di lavoro al variare delle esigenze, evitando di cristallizzare progettisti su progetti specifici anche quando questi siano in una fase di assestamento finale che non richiede un grosso carico di lavoro.

Un terzo aspetto rilevante in relazione alla documentazione per la manutenzione è quello della formazione di personale nuovo. Di fatto l'acquisizione di competenza tecnica negli ambienti di produzione di software avviene oggi prevalentemente attraverso meccanismi di imitazione dell'ambiente e di perpetuazione di tradizioni che hanno le loro radici nel fatto che il personale neoassunto viene spesso dedicato a maintenance di programmi. In tal modo una cattiva documentazione di manutenzione può ridurre le possibilità di crescita del nuovo personale, demotivarlo, e costituire un grosso freno a cambiamenti della cultura tecnica di un ambiente di produzione.

3. Difficoltà nella costruzione della documentazione per la manutenzione.

La documentazione per la manutenzione viene spesso, per non dire esclusivamente, scritta quando ormai il prodotto è stato completato: in tal modo essa non ha più alcuna possibilità di fungere da strumento di riflessione critica che possa influenzare la stesura dei programmi. Proprio per questo motivo il suo ruolo viene identificato fondamentalmente solo nel permettere correzioni ai programmi, e viene vissuta dal progettista che la deve scrivere come un compito fastidioso, come una vera e propria fase di lavoro successiva alle precedenti, in cui le sue capacità tecniche non vengono utilizzate.

Per tali ragioni la documentazione di manutenzione è la prima a perdere di qualità, e quindi di efficacia, in caso di problemi di pianificazione, di ritardi, di esigenza di forza di lavoro: ciò è ancora più misurabile e generalizzato nel caso di correzioni ai programmi, per cui spesso non viene riportata la corrispondente correzione nella documentazione.

4. Una proposta di soluzione

Le richieste che si possono fare ad uno standard di documentazione di manutenzione e all'organizzazione della costruzione di tale documentazione possono essere quindi sintetizzate come segue: è necessario che la documentazione accompagni la costruzione dei programmi e sia in grado di influenzarla, che essa sia sempre aggiornata, che permetta un rapido inserimento di persone nuove su un progetto non ancora completo (e quindi cresca in modo sincrono ai programmi) e che possa essere di guida per la formazione di personale di esperienza limitata.

Tali richieste possono essere soddisfatte cercando di integrare la documentazione nel programma stesso, in modo che la sua lista di compilazione costituisca la parte più rilevante della documentazione. La qualità della documentazione, la mobilità del personale sui progetti, la crescita di personale nuovo saranno in tal caso funzione della qualità del programma.

Uno degli aspetti fondamentali di una tale documentazione deve essere quello di "guidare" il lettore alla comprensione di programmi anche di rilevanti dimensioni, per gradi, con un rispetto della propedeuticità della presentazione di nuove informazioni: sarà cioè necessario che la documentazione, ovvero il programma, sia suddivisa in moduli organizzati in modo gerarchico, il più elevato dei quali descriva l'intero programma in termini di moduli di livello inferiore, e così via, fino ai livelli elementari. Si richiede cioè l'adozione di tecniche di programmazione strutturata top-down , con una precisa organizzazione gerarchica del programma in moduli.
Gli strumenti linguistici impiegati nella costruzione della lista di compilazione autodo-

cumentata non potranno quindi che essere lo stesso linguaggio di programmazione, accompagnato da un uso adeguato di commenti: uno standard di organizzazione del programma, dell'uso di commenti e quindi dell'organizzazione della lista è in grado di determinare in modo completo la costruzione di programmi autodocumentati, rendendo necessariamente molto aderente la documentazione del programma alla sua reale struttura (anche nel caso di modifiche locali successive,che maggiormente invogliano all'alterazione degli opportuni commenti, rispetto al meccanismo tradizionale, che richiede una procedura completamente differente e separata per la modifica dei documenti cartacei, anche dal punto di vista puramente burocratico).

5. Le caratteristiche della proposta

Per la buona descrizione "didattica" della struttura e delle caratteristiche di dettaglio di un programma è necessario fornire :

a. una testata che contenga le "specifiche di targa" per individuare il prodotto nell'ambito del sistema;

b. indicazioni sulla struttura globale del prodotto, in termini della sua organizzazione gerarchica: tali indicazioni possono essere direttamente fornite sulla lista del programma mediante la descrizione dell'albero che rappresenta la gerarchia del programma, effettuata mediante commenti;

c. la descrizione dei dati, in particolare di ingresso e uscita, e le aree correlate,effettuata direttamente mediante le istruzioni del linguaggio di programmazione, opportunamente commentate;

d. la descrizione delle diverse porzioni del programma , di diverso livello gerarchico, inquadrando, per ciascuna porzione, le sue caratteristiche dal punto di vista funzionale e le sue caratteristiche da un punto di vista fisico, in particolare in relazione agli aspetti che maggiormente interessano interfacce con altri moduli.

   1. nome del modulo;
   2. una descrizione della trasformazione che il modulo effettua, da un punto di vista puramente funzionale, con la specificazione degli obiettivi che tale trasformazione si propone di raggiungere;
   3. le condizioni di ingresso, ovvero
      - descrizione dei dati in ingresso al modulo, dal punto di vista del loro significato in termini di problema, cioè illustrando le entità o le informazioni che essi rappresentano;
      - descrizione delle aree di memoria corrispondenti ai dati di ingresso, dei relativi valori possibili o reali, con riferimento ai moduli da cui provengono i dati;
   4. una descrizione dell'algoritmo impiegato per effettuare la trasformazione, eventualmente coinvolgente i nomi dei dati descritti;
   5. le condizioni di uscita, in modo analogo alle condizioni di ingresso;
   6. le aree di lavoro, cioè una descrizione di tutte quelle aree che non essendo nè di ingresso nè di uscita, vengono usate all'interno del modulo: spesso in esse si trovano le interfacce con i moduli richiamati;
   7. l'elenco dei moduli richiamati da quello in esame, eventualmente corredato da una brevissima descrizione della funzione svolta da ogni modulo;
   8. le interfacce dati con tali moduli, cioè l'elenco delle aree interessate nella comunicazione tra i moduli, corredato eventualmente di valori possibili e del relativo significato;
   9. i trasferimenti di controllo ("da"e "a") non organizzati in modo gerarchico (per esempio passaggio da una sezione del programma ad una successiva);
   10. note aggiuntive che il programmatore ritiene utili alla comprensione del programma.

Per ciascun modulo, alla parte di listing descritta qui sopra deve seguire il codice che realizza il modulo stesso, di dimensioni tali da raggiungere, con la parte di descrizione illustrata, la dimensione media di due pagine di listing, al fine di mantenere snello il lavoro di lettura e consultazione.

Il metodo di autodocumentazione prevede l'inserimento guidato di commenti intermedi inseriti nel codice stesso, classificati secondo le due fondamentali categorie:

- motivazioni, cioè descrizioni delle intenzioni e degli obiettivi di una certa porzione di codice;
- asserzioni, cioè descrizioni dello stato della memoria ad un certo istante dell'elaborazione.

Il risultato della proposta individua due strutture di moduli di codifica che supportino l'inserimento di tali informazioni nella lista del programma: di essi il primo (vedi fig.1) è unico per l'intero programma, ed ha lo scopo di contenere le informazioni di targa e la descrizione globale della struttura gerarchica del programma mediante un albero; il secondo (vedi fig.2) è presente all'apertura di ogni modulo di qualunque livello.

Le figg.3,4,5 forniscono un esempio di applicazione dell'autodocumentazione a un programma in assembler. Sono mostrati la testata del programma e la descrizione, più relativo codice, del modulo di primo livello manca la descrizione dei dati, scritta nel linguaggio di programmazione, con l'aggiunta di commenti, su normali moduli.

6. Conclusioni

Lo standard proposto è largamente indipendente dal linguaggio; per esso è in corso una applicazione nei laboratori di Pregnana per il linguaggio assembler NAL, per cui sono stati messi a punto moduli di codifica prestampati (vedi figg. 1 e 2).

Esperimentazioni condotte su programmi Cobol hanno dato ottimi risultati, soprattutto in relazione all'inserimento estremamente semplice di nuovo personale in un progetto in corso.

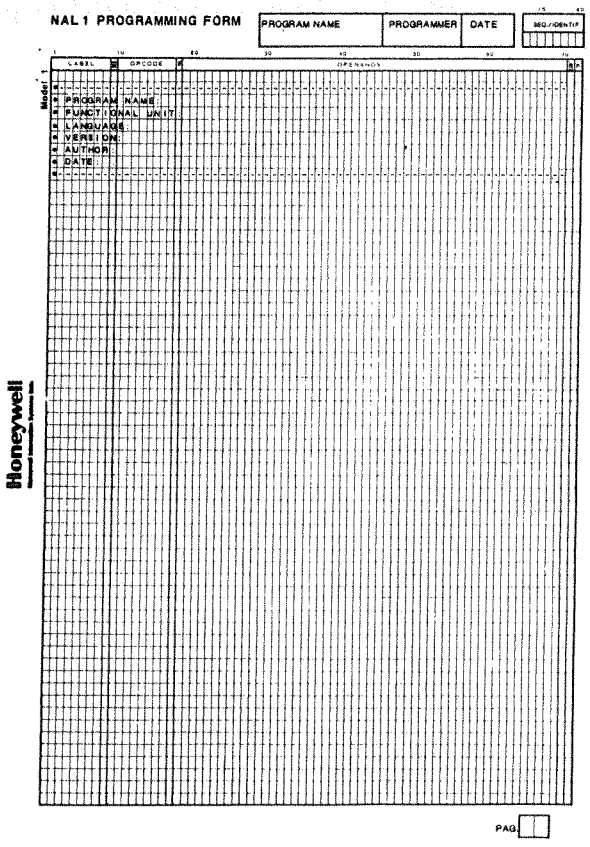


Fig. 1

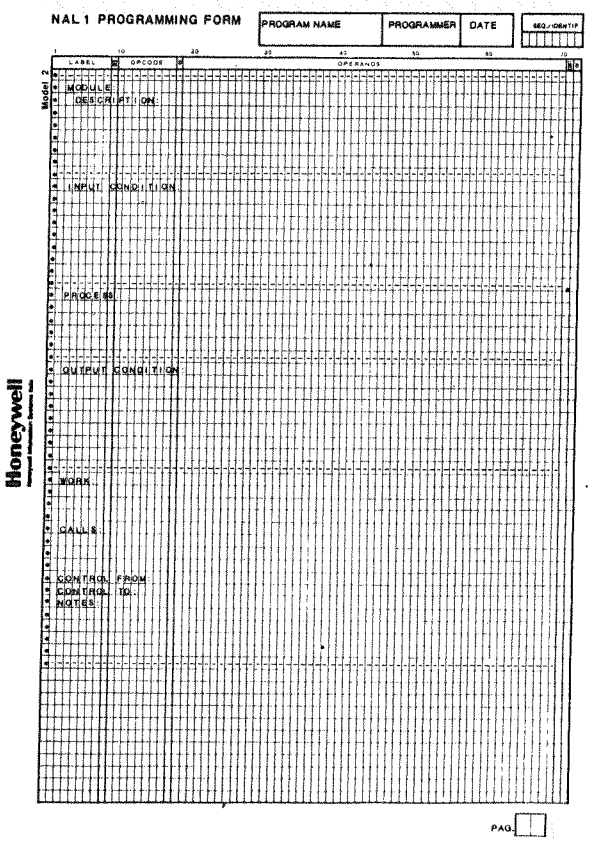


Fig. 2

```
*-----------------------------------------------------------------------
* PROGRAM NAME:    MECGROW
* FUNCTIONAL UNIT:MECCA
* LANGUAGE:        STNAL + NAL
* VERSION:         0
* AUTHOR:          GOBBI, TRABATTONI
* DATE:            1976
*-----------------------------------------------------------------------
*
*                 STRUCTURE OF MECGROW
*
*
*
*                   .....INIT
*                   .
*                   .
*                   .                                   ....TESTCRNO
*                   .                                   .
*                   .   .....CARDJOB....CRUPDATE.........COPYSUBF  R
*                   .   .                               .
*                   .   .       0           *           ....COPYFSUB  R
*                   .   .
*                   .   .
*        MECGROW.....  *
*                   .   .
*                   .   .                               ....TESTCONO
*                   .   .                               .
*                   .   .....CONSJOB....CONSUPD.........COPYSUBF  R
*                   .                                   .
*                   .           0           *           ....COPYFSUB  R
*                   .
*                   .
*                   .
*                   .....TERM
*
*
*                 O=SELECTION     *=ITERATION
*                 R=MODULE USED IN SEVERAL PARTS OF THE PROGRAM
```

Fig. 3

```
*-----------------------------------------------------------------------
* MODULE: MECGROW
*  DESCRIPTION: HANDLING OF A DEVELOPMENT SUPPORT LIBRARY FOR COBOL
*               PROGRAMS.  ADDITIONS, CHANGES AND DELETIONS ARE ACCEPT-
*               ED BOTH FROM SEQUENTIAL FILE AND FROM CONSOLE.
*-----------------------------------------------------------------------
* INPUT CONDITION:
*   SULIB :[QUEUED LINKED LIBRARY OF COBOL SOURCE PROGRAMS]
*   INFILE:[SEQUENTIAL FILE OF UPDATINGS]
*   [CONSOLE UPDATINGS, IF ANY]
*-----------------------------------------------------------------------
* PROCESS:OPEN AND CHECK FILES; REPEAT THE FOLLOWING OPERATIONS UNTIL
*   LIB.ERROR OR END OF UPDIN: REPEAT LIB.HANDLING WITH UPDATINGS FROM
*   INFILE UNTIL ERROR OR END OF INFILE OR SWITCH TO CONSOLE INPUT,
*   REPEAT LIB.HANDLING FROM CONSOLE UNTIL LIB.EPPOR OR SWITCH TO
*   INFILE INPUT; CLOSE FILES.
*-----------------------------------------------------------------------
* OUTPUT CONDITION:
*   SULIB :[UPDATED WHENEVER POSSIBLE]
*   [CONSOLE MESSAGES IN CASE OF ERRORS]
*-----------------------------------------------------------------------
* WORK: EOF :[END OF FILE VALUE OF G4 = X"F1F0"]
*       UPDINHEAD :[FIRST 6 CHAR. OF INFILE RECORD AREA]
*       CONSINHD  :[FIRST 6 CHAR. OF CONSOLE AREA]
*       STOP :[SWITCH COMMAND = C"STOP  "]
*       WORKIN :[SEQUENTIAL WORK FILE]
*       WORKOUT :[SEQUENTIAL WORK FILE]
*       LIBFLAG :[FLAG OF SULIB I/O ERROR = C"1"]
* CALLS: INIT :[FILE OPEN AND CHECK]
*        CARDJOB :[LIBRARY HANDLING FROM INFILE]
*        CONSJOB :[LIBRARY HANDLING FROM CONSOLE]
*        TERM :[FILE CLOSE]
* CONTROL FROM: OPERATING SYSTEM
* CONTROL TO: OPERATING SYSTEM
* NOTES: UPDATING OPERATIONS START AND END WITH INPUT FROM INFILE.
*        SEE FUNCTIONAL SPECS OF MECGROW FOR COMMANDS, DATA FORMAT,
*        ERROR RECOVERY AND MESSAGES, EXAMPLES OF OPERATION.
*-----------------------------------------------------------------------
```

Fig. 4

```
*M        [OPEN AND CHECK FILES];
          BBICR     B2,INIT
*
*M        [HANDLE LIBRARY OF COBOL PROGRAMS];
 $REPEAT;
*
*M             [HANDLE LIBRARY WITH INPUT FROM INFILE];
 $REPEAT;
          CALL      CARDJOB
*M             [EXIT IF END OF INFILE OR SWITCH TO CONSOLE OR LIB.ERR.];
 $EXIT CMP=(CG2,'G4,EOF',EQ);
 $EXIT CMP=(CCE,'UPDINHD,STOP',EQ);
 $EXIT CMP=(CI,'LIBFLAG,C"1"',EQ);
 $ENDREP;
*A             :[SULIB UPDATED AND/OR SWITCH TO CONSOLE OR LIB.ERROR]
*
 $SIF CMP=(CI,'LIBFLAG,C"1"',EQ),THEN;
 $WRITE P1='I/O ERROR ON LIBRARY. END OF MEGROW',DEV=CTW;
 $ENDIF;
*
*M        [EXIT IF END OF INFILE OR LIBRARY ERROR];
 $EXIT CMP=(CI,'LIBFLAG,C"1"',EQ);
 $EXIT CMP=(CG2,'G4,EOF',EQ);
*
*             [HANDLE LIBRARY WITH INPUT FROM CONSOLE];
 $REPEAT;
          CALL      CONSJOB
*M             [EXIT IF SWITCH TO INFILE OR LIBRARY ERROR];
 $EXIT CMP=(CCE,'CONSINHD,STOP',EQ);
 $EXIT CMP=(CI,'LIBFLAG,C"1"',EQ);
 $ENDREP;
*A             :[SULIB UPDATED AND/OR SWITCH TO INFILE OR LIB.ERROR]
*
 $SIF CMP=(CI,'LIBFLAG,C"1"',EQ),THEN;
 $WRITE P1='I/O ERROR ON LIBRARY. END OF MECGROW',DEV=CTW;
 $ENDIF;
*
*M        [EXIT IF LIBRARY ERROR];
 $EXIT CMP=(CI,'LIBFLAG,C"1"',EQ);
 $ENDREP;
*A        [SULIB UPDATED AS REQUESTED OR ERR.MESSAGES DISPLAYED]
*
*M        [CLOSE FILES];
          BBICR     B2,TERM
          EXIT
```

Fig. 5

Un programma di 13.000 schede (7.000 commenti) è stato realizzato da due persone fisse più 4 persone presenti una alla volta per 3-4 settimane. L'addestramento consisteva nella lettura delle specifiche funzionali, in un seminario di tre ore e nelle specifiche di ingresso-trasformazione-uscita delle parti da implementare. La lettura delle parti già fatte garantiva l'uniformità di qualità e stile. La correzione di errori nel prodotto distribuito ha richiesto nella maggior parte dei casi pochi minuti, non più di un'ora nei casi peggiori.
Pertanto la portabilità uomo (trasferibilità di persone sul progetto), visti i bassi costi di addestramento e di manutenzione, si è rivelata ottima.

7. Riferimenti bibliografici

1. G. Degli Antoni, G. Gobbi
"Aspetti relativi alla documentazione dei programmi" - Giornate di studio su "Programmazione strutturata: esperienze e orientamenti" - HONEYWELL I.S.I.- Milano, 23,24,25 giugno 1976.

2. M. Maiocchi
"Documentazione di disegno dei programmi e autodocumentazione" HONEYWELL I.S.I. - Documento interno n. A78119847.

3. G. Gobbi, M. Maiocchi, A. Vignoni, F. Vitale
"Documentazione di disegno dei programmi: una proposta per autodocumentazione e documentazione durante lo sviluppo dei programmi" - HONEYWELL I.S.I. - Documento interno, aprile 1977.

4. C. Barassi
"Programmazione strutturata. Rinnovamento culturale e crisi di rigetto" - Giornate di studio su "Programmazione strutturata: esperienze e orientamenti" - HONEYWELL I.S.I. - Milano, 23, 24, 25 giugno 1976.

5. R. L. Patrick
Software Engineering and Life Cycle Planning - Datamation, 22, 12, 1976.

DOCUMENTAZIONE PER L'UTENTE : ALCUNE INDICAZIONI E UN ESEMPIO

A. Maserati(o), S. Mazzocchi(o), R. Polillo(o), G. Torriani(+)

1. INTRODUZIONE

Lo scopo di questa nota è quello di proporre alcune indicazioni di metodo per la costruzione di "manuali per l'utente" relativi a prodotti software. Tali considerazioni sono state sperimentate nella costruzione del manuale di TPS (Transactional Program Sistem) del Livello 62. [1].

Dopo aver ricordato che esistono differenti tipi di documenti diretti all'"utente" di un prodotto software (par.2) si accenna ai problemi che rendono complessa l'attività di costruzione di documentazione (par.3), e alle conseguenze di una cattiva documentazione per l'utente (par.4). Si delinea quindi (par.5) una serie di linee guida per la costruzione di manuali utente "facili da usare", e si descrive il semplice standard proposto (par.6). Si mostra, infine, come un manuale così concepito possa essere usato nell'attività di formazione (par.7), e si riassumono gli aspetti fondamentali della proposta (par.8).

La metodologia proposta è illustrata attraverso esempi tratti dal manuale di TPS citato.

2. LA DOCUMENTAZIONE PER L'UTENTE

Il termine "documentazione per l'utente", se da un lato permette di individuare abbastanza chiaramente a che cosa non si riferisce (ad esempio, la documentazione per l'utente di un prodotto software si contrappone a quella di disegno, o a quella di manutenzione [2]), non caratterizza tuttavia un'unica classe di prodotti.

(o) Università di Milano - Istituto di di Cibernetica e H.I.S.I.
(+) H.I.S.I.

Infatti, gli "utenti" (almeno in una accezione non restrittiva del termine) di un prodotto software sono in generale diversi (ad esempio, DP managers, programmatori, istruttori, end-users...), e diversa è la visibilità che tali utenti necessitano sui prodotti. D'altro canto, anche considerando una classe "omogenea" di utenti, gli scopi a cui tale documentazione è rivolta possono essere molteplici.

Non è negli scopi di questa nota proporre una classificazione della letteratura esistente attorno a prodotti software e che possa essere considerata come "documentazione per l'utente". Ci limiteremo a ricordare alcuni tipi di documenti che, anche se non sempre disponibili, posseggono una collocazione abbastanza precisa nelle esigenze dell'utenza, e la cui funzione non può essere trascurata da chi si accinga a costruire della documentazione per l'utente.

Presentazioni, pieghevoli, ecc. Si tratta di documenti sintetici, che caratterizzano, in genere, l'ambito del prodotto illustrato (in che ambiente H/W, S/W e applicativo esso si colloca), il suo scopo (i vantaggi offerti all'utente dalla sua adozione), la classe di funzionalità supportate, ecc.

Overviews, descrizioni generali, introduzioni a ..., ecc. In questa classe collochiamo quei documenti, quasi sempre disponibili a livello di sistema, meno spesso a livello di singoli componenti, il cui scopo è quello di fornire una classificazione organica delle principali caratteristiche funzionali di un prodotto, e di illustrare i concetti generali ad esso connessi.
Tali documenti, che possono prescindere dagli aspetti immediati d'uso del prodotto (sintassi dei comandi, ecc.) hanno lo scopo, da un lato, di fornire la struttura concettuale necessaria per comprendere la filosofia

del prodotto (architettura e sue motivazioni, funzionalità offerte, soluzioni adottate, terminologia,...). Dall'altro, essi permettono al lettore interessato di collocare il prodotto nell'ambito di prodotti consimili.

User guides. Con questo termine vengono indicati quei documenti che hanno lo scopo di fare acquisire, attraverso un uso graduale di esempi commentati, familiarità del prodotto a chi lo userà direttamente (ad esempio, il programmatore nel caso in cui il prodotto sia un linguaggio di programmazione).
L'enfasi sarà pertanto posta sulle modalità di uso più frequenti (o, in qualche senso, migliori o suggerite) del prodotto. Non verranno invece enfatizzati aspetti di dettaglio o situazioni di carattere eccezionale.
Questa letteratura viene a volte fornita nella forma di testi per autoistruzione.

Reference manuals. Sono i documenti di consultazione per l'uso del prodotto. L'enfasi sarà pertanto posta sulla completezza dell'informazione, e questa sarà organizzata in modo tale da facilitare il reperimento di aspetti di dettaglio. Essendo privilegiato l'aspetto consultativo rispetto a quello didattico, che richiede una presentazione graduale ed esemplificata delle varie caratteristiche del prodotto, questi manuali si rivelano molto spesso inadatti a quei lettori che non ne conoscano già le caratteristiche principali.

Reference cards, vademecum, pocket guides, ecc. Sono dei promemoria compatti e maneggevoli (spesso in forma di tavole sinottiche su cartoncini tascabili) che riportano tutte le informazioni di dettaglio necessarie all'uso di un prodotto: sintassi dei comandi, procedure da effettuare, mappe di memoria, tracciati records, tabelle, ecc.
Presuppongono nell'utente padronanza del prodotto (e disponibilità del reference manual).

eccetera.

3. PROBLEMI NELLA COSTRUZIONE DI DOCUMENTAZIONE PER L'UTENZA

Già gli esempi indicati al paragrafo precedente permettono di comprendere come la costruzione di documentazione per l'utenza sia un'attività complessa, non soltanto per la complessità intrinseca dei prodotti software che tale documentazione deve rendere accessibili, ma anche perchè le esigenze dell'utenza possono essere diversificate, e non sempre preconfigurabili con facilità.

Ciò comporta rischi non trascurabili per il costruttore di prodotti software. Da un lato, quello di non fornire all'utente tutta la documentazione di cui necessita, limitandosi, ad esempio, a produrre soltanto manuali di riferimento, e trascurando o sottodimensionando il materiale rivolto alla formazione o alla sintesi. Dall'altro, quello di produrre un eccessivo numero di documenti con finalità diverse sullo stesso prodotto, con conseguente possibilità di ridondanze o incongruenze, e comunque con alti costi di sviluppo e di manutenzione.

I problemi di cui sopra sono ulteriormente esasperati dal fatto che, in un ambiente di produzione di software, l'attività di documentazione in generale - e quella di documentazione per l'utenza in particolare - viene spesso percepita come marginale rispetto all'attività di sviluppo in senso tradizionale. In ogni caso, ad essa raramente viene riconosciuto quell'alto grado di professionalità che le è proprio. In effetti, occorre riconoscere che la costruzione di "buona" documentazione per l'utenza è compito difficile. Da un lato, infatti, essa richiede competenze tecniche di tipo sistemistico e ingegneristico non superficiali (chi documenta deve poter comunicare di frequente e con competenza tecnica con chi sviluppa e con chi pianifica [3]); dall'altro, essa richiede esperienza e, soprattutto, alta sensibilità sui problemi dell'utenza. Tale sensibilità è spesso non facile da reperire in personale con orientamento sistemistico. Di fatto, un atteggiamento di tipo sistemistico e un atteggiamento orientato all'utenza portano spesso ad attribuire un peso differente ai vari aspetti di un prodot-

to software, e, di conseguenza, comportano una differente organizzazione del manuale che lo descrive, o l'attribuzione di un differente peso specifico alle sue varie parti. Non è raro, ad esempio, il caso in cui la cattiva documentazione di un aspetto di dettaglio dal punto di vista dello sviluppatore produca difficoltà molto serie nell'uso di un prodotto (°).

Infine, occorre non dimenticare che un manuale è un prodotto esso stesso, il cui successo (strettamente legato al successo del prodotto da esso documentato) dipenderà in larga misura dalla semplicità del suo uso. Pertanto, la costruzione di "buoni" manuali per l'utenza richiederà anche, e soprattutto, esperienza e sensibilità su aspetti didattici e di "presentazione dell'informazione". Quest'attitudine, molto spesso trascurata, non può evidentemente essere disgiunta dalle due attività precedenti. Un manuale carente sotto questo aspetto sarà difficilmente migliorabile, se non in modo superficiale, da un intervento "a posteriori", che ne curi esclusivamente la "presentazione": questa andrà infatti studiata caso per caso, anche se nell'ambito di standards precostituiti, sulla base delle caratteristiche funzionali del prodotto e sulla base del suo uso.

(°) Si pensi, tanto per fare un esempio macroscopico, al diverso atteggiamento di chi sviluppa un prodotto e di chi lo usa nei confronti di una macro di JCL che serva a richiamarne l'esecuzione. Per lo sviluppatore, essa sarà un aspetto di dettaglio (infatti, la sua complessità non sarà in genere paragonabile a quella del prodotto, sarà spesso sviluppata alla fine, ecc.). Per l'utente, essa costituirà, in un certo senso, l'aspetto fondamentale: senza una chiara padronanza dei suoi parametri egli non sarà - semplicemente - in grado di utilizzare il prodotto.

4. CONSEGUENZE DI UNA CATTIVA DOCUMENTAZIONE PER L'UTENTE

Un prodotto software diviene accessibile all'utente attraverso la sua documentazione: senza di questa, esso sarà indefinito, e pertanto inutilizzabile. La "qualità" di un prodotto software, dal punto di vista della utenza, non può, pertanto, essere valutata indipendentemente dalla qualità della sua documentazione. Una cattiva documentazione potrà pregiudicare completamente il successo di un prodotto presso l'utenza, indipendentemente dalle sue qualità intrinseche (facilità d'uso, architettura, affidabilità, ecc.). Conversamente, una "buona" documentazione per l'utenza potrà equilibrare, almeno in parte, le carenze di un prodotto concepito in modo insoddisfacente.

Quanto sopra è tanto più evidente quando si pensi che la qualità della documentazione è un fattore legato in modo non trascurabile ad aspetti complessivi di organizzazione del lavoro [4].
Una cattiva documentazione comporta infatti maggiori costi di formazione, minore mobilità di personale sui progetti, minore consapevolezza dei ruoli individuali nella attività complessiva. In una parola, minore produttività presso l'utenza.

D'altra parte, in un momento in cui la evoluzione dei sistemi di elaborazione porta a riconoscere il ruolo sempre più centrale dell'utente finale, e a enfatizzare gli aspetti di semplicità d'uso o gli "human factors", la qualità della documentazione per l'utenza assume un ruolo sempre più decisivo per il successo commerciale di un prodotto.

5. DOCUMENTAZIONE "FACILE DA USARE"

Per quanto detto più sopra, costruire della buona documentazione per l'utenza significa, in primo luogo, costruire della documentazione "facile da usare". Questo slogan, se applicato coerentemente, comporta numerose implicazioni, che impongono un ripensamento sia delle usuali forme di presentazione dell'informazione (grafica, impaginazione, strutturazione in sottoparti

ecc.), sia delle tecniche di organizzazione degli argomenti, sia, infine, del numero e tipo di manuali associati a un prodotto.

In primo luogo, le difficoltà segnalate nel paragrafo 3 relativamente allo sviluppo, uso e manutenzione di una molteplicità di manuali diversi, orientati a differenti scopi, suggeriscono la costruzione di un documento unico, in grado di soddisfare, per quanto possibile, alle esigenze di :

- chi desideri avere una visione sintetica di insieme del prodotto
- chi desideri apprenderne (gradualmente) l'uso
- chi debba effettivamente usare il prodotto.

Ciò comporta, in sostanza, che tale manuale dovrà assolvere alle varie funzioni individuate nel paragrafo 2: dovrà cioè potersi usare, di volta in volta, come presentazione del prodotto, come sua descrizione generale, come guida all'utente inesperto, come manuale di riferimento e, infine, come "vademecum" durante l'uso del prodotto stesso.

Per quanto riguarda la presentazione dell'informazione, varie tecniche sono evidentemente possibili. L'esigenza di una comunicazione efficace e sintetica suggerisce, innanzitutto, di affiancare all'usuale strumento comunicativo costituito dal testo scritto altri strumenti, e in primo luogo la immagine. Questa potrà essere usata non soltanto, come tradizionalmente avviene, a corredo del testo scritto (figure esplicative richiamate nel testo, tabelle, grafici, ecc.), ma addirittura come veicolo primario dell'informazione, di cui il testo scritto è complemento, o appendice.
Altre tecniche comunicative saranno invece studiate per facilitare la selezione o l'accesso all'informazione desiderata, sia essa in forma di immagine o in forma di testo: uso di pagine di colore o di consistenza differenti, possibilmente estraibili, uso dell'indentazione nel testo, caratteri speciali (maiuscole o neretti), indici o riferimenti ad altre parti del manuale.
Più oltre verrà illustrato uno standard dotato di particolari caratteristiche di semplicità e di efficacia, adottato nella costruzione del manuale di TPS.

Per quanto riguarda l'organizzazione degli argomenti, tre linee guida ci sembrano rilevanti:

- raggruppamento delle informazioni sulla base dei loro utenti (o del loro uso).
In sostanza, tutte le informazioni rivolte a un particolare utente, o necessarie per una attività bene individuata, sono fisicamente vicine, raggruppate in una sezione diretta a quello specifico utente (o uso). Così, dall'indice stesso del manuale si individueranno, in primo luogo, i diversi utenti del prodotto. Questi potranno, almeno in linea di principio, utilizzare solo la porzione di manuale a loro specificamente diretta e potranno, durante una specifica attività, focalizzare la loro attenzione su un numero limitato di pagine, fra loro fisicamente contigue. Una o più sezioni introduttive, contenenti la descrizione dell'ambito, dello scopo e delle principali classi di funzionalità offerte dal prodotto permetteranno invece di collocare con precisione le funzionalità del prodotto in relazione alle diverse attività implicate e ai suoi diversi utenti, e forniranno a questi una visione e un linguaggio comune sul prodotto.
Ad esempio, la fig.1 mostra l'organizzazione delle varie sezioni del manuale di TPS.

- organizzazione gerarchica delle informazioni. All'interno di una singola sezione del manuale, l'informazione viene presentata per successivi livelli di dettaglio, dagli aspetti di insieme fino a quelli più specifici. I vantaggi di un'organizzazione di questo tipo possono essere così sintetizzati :

  - chiarezza dell'organizzazione degli argomenti, con conseguente possibilità di localizzare immediatamente l'informazione desiderata attraverso l'alberatura degli argomenti stessi ("accesso diretto" all'informazione);
  - possibilità di fermarsi al livello di dettaglio desiderato, senza essere "disturbati" da informazioni troppo specifiche

1. INTRODUCTION
(una breve presentazione, per chi desideri conoscere ambito, scopo e classi di funzionalità offerte dal prodotto)

2. TPS MAIN FEATURES
(concetti generali, non immediatamente legati all'uso del prodotto, per chi desideri conoscerne le caratteristiche rilevanti)

3. TERMINAL OPERATOR MANUAL
(tutto quanto serve all'operatore al terminale per la sua attività. Le sezioni 1 e 2 non sono prerequisito necessario, anche se suggerito)

4. SYSTEM OPERATOR MANUAL
(tutto quanto serve all'operatore di console per la sua attività. Le sezioni 1 e 2 non sono prerequisito necessario, anche se suggerito)

5. PROGRAMMER MANUAL
(tutto quanto serve a chi sviluppa la applicazione transazionale. Prerequisiti: sezioni 1 e 2. Contiene sottosezioni volte a specifiche attività: come codificare un'applicazione, come compilarla, come lanciarla, come provarla).

fig. 1

rispetto alle particolari esigenze ("discesa" nell'albero degli argomenti fino al livello desiderato);

- possibilità di collocare l'informazione desiderata in un ambito più vasto ("risalita" nell'albero degli argomenti a partire dall'informazione desiderata).

La figura 2 mostra un esempio (tratto dal manuale TPS) di tre "moduli" consecutivi (vedi par.6), corrispondenti a tre diversi livelli di dettaglio di uno stesso argomento.

- Concentrazione di tutte le informazioni di dettaglio riguardanti una medesima attività. In tal modo, il manuale assolve anche alle funzioni di "reference card" o di "vademecum" (cfr.par.2) : tutte le informazioni di dettaglio su un certo aspetto d'uso (ad es., la sintassi dei comandi, ecc.) vengono concentrate su una singola pagina, colorata (in modo da essere riconoscibile), estraibile, su cartoncino ("reference page").
La fig.3 mostra un esempio di "reference page" tratta dal manuale di TPS.

6. PRESENTAZIONE DELL'INFORMAZIONE

Descriviamo ora brevemente lo standard proposto.

Un manuale è organizzato in sezioni, ognuna delle quali:

- inizia con una pagina che ne contiene (soltanto) il titolo, stampata su cartoncino in modo da rendere agevole la individuazione della sezione;
- ha una numerazione indipendente (sigla che individua la sezione, numero di pagina);
- è costituita da "moduli PSPN" e, se necessario, da pagine di riferimento.

6.1. I moduli PSPN

Ogni modulo "PSPN" ("pagina-sì-pagina-no")(fig.4), che sviluppa un "singolo" argomento, possiede una struttura fissa:

- la pagina sinistra ("pagina-sì") (altri tre esempi in fig.2) è costituita da una immagine che sintetizza i principali contenuti del modulo. L'immagine è racchiusa da una cornice standard, interrotta dal titolo (obbligatorio) del modulo.
Le dimensioni della cornice (17,5x21,5 cm.) e dei caratteri sono studiati in modo tale che la pagina sinistra sia utilizzabile come "visual" per lavagna luminosa, una volta fotocopiata su trasparenti.

- la pagina destra ("pagina-no"), in testa alla quale è riportato lo stesso titolo della pagina sinistra, contiene il testo del modulo, che illustra e completa i contenuti del "visual".
Il testo è organizzato su due livelli di indentazione, corrispondenti agli aspetti principali o di dettaglio.

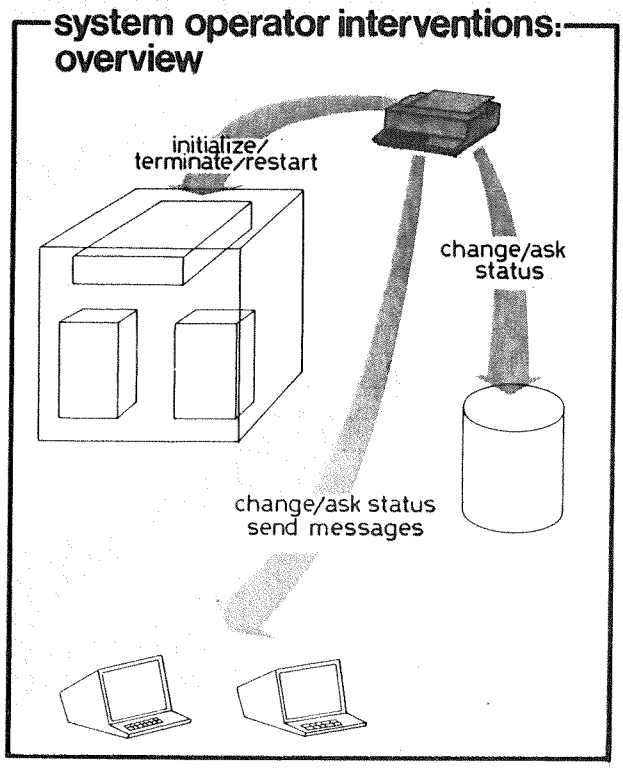


Fig. 2a

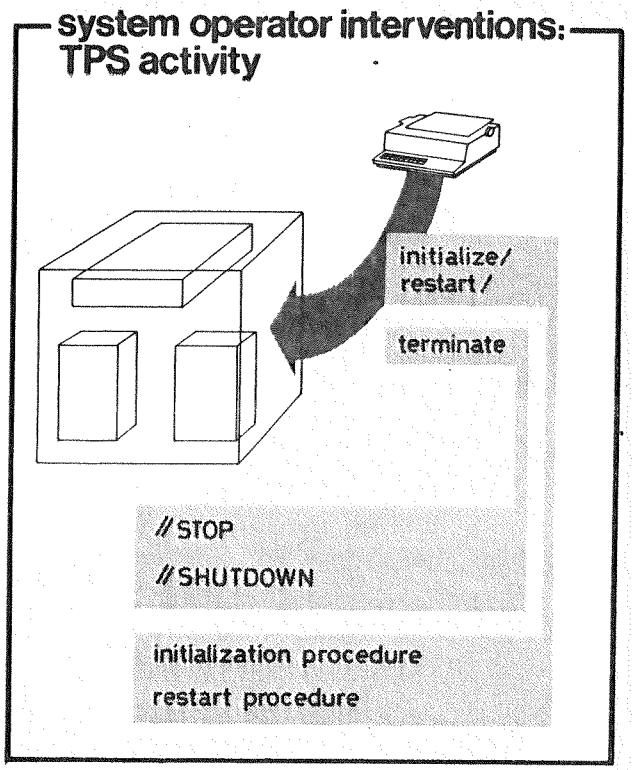


Fig. 2b

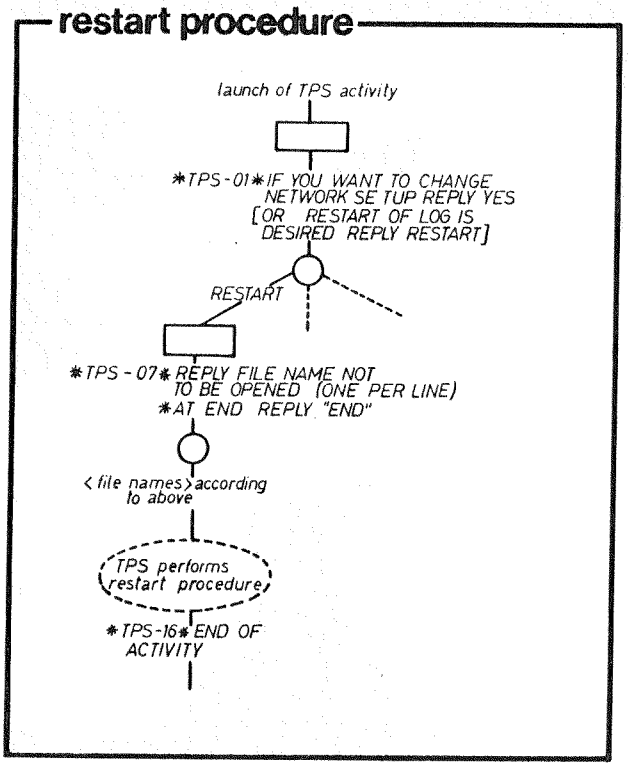


Fig. 2c

Fig. 2a, 2b, 2c,: tre esempi di "pagine sinistre" (cfr. par. 6), o diagrammi che presentano uno stesso argomento a diversi livelli di dettaglio. La fig. 2a (livello alto) schematizza le varie classi di interventi effettuabili da parte dell'operatore di consolle durante la esecuzione di un'applicazione transazionale (sull'applicazione in esecuzione, schematizzata da un monitor e da due routines transazionali presenti in memoria, sui files in linea, sui terminali coinvolti). La fig. 2b (livello medio) dettaglia gli interventi possibili sull'applicazione in esecuzione, mostrando la sintassi di alcuni comandi o la esistenza di specifiche procedure da effettuare. La fig. 2c (livello basso) specifica la procedura di restart.

Sottotitoli o neretti evidenziano ulteriormente la struttura del discorso.

La pagina sinistra e la pagina destra di un modulo sono pensate per essere lette, almeno in linea di principio, "in parallelo": non esiste, in altre parole, una sequenza preordinata di lettura. La pagina sinistra costituisce un supporto all'informazione contenuta nella pagina destra: essa ha, in sostanza,le stesse funzioni di un "visual" per lavagna luminosa. Tali funzioni, nei dettagli, saranno studiate di volta in volta, a seconda delle esigenze del particolare argomento. Tipicamente, il "visual" conterrà un riassunto per punti dell'informazione del modulo, oppure l'elenco dei temi sviluppati, eventualmente collocati nel loro contesto. Altre volte mostrerà l'oggetto del discorso svolto a destra (ad es., un brano di programma commentato nella pagina destra, la sintassi di un comando..), oppure ancora un diagramma esplicativo, ecc. In ogni caso, la pagina sinistra faciliterà la consultazione del manuale, permettendo di localizzare con facilità, senza leggere il testo, l'informazione desiderata. La pagina destra, invece, inizierà con una breve introduzione all'argomento del modulo, dirigendo implicitamente l'attenzione del lettore al "visual".Ne esporrà quindi i contenuti in forma organica e strutturata (spazi, neretti, titoli, indentazione), per facilitare l'accesso all'informazione di dettaglio. Si concluderà, se necessario, con un brano di conclusioni e di collegamento con il modulo successivo.

Il primo modulo di ogni sezione conterrà indicazioni per la sua lettura: la pagina sinistra riporterà l'indice della sezione, mentre quella destra ne individuerà lo scopo, la classe di utenti a cui è diretta, i legami con altre sezioni e ne motiverà, se necessario, l'indice.

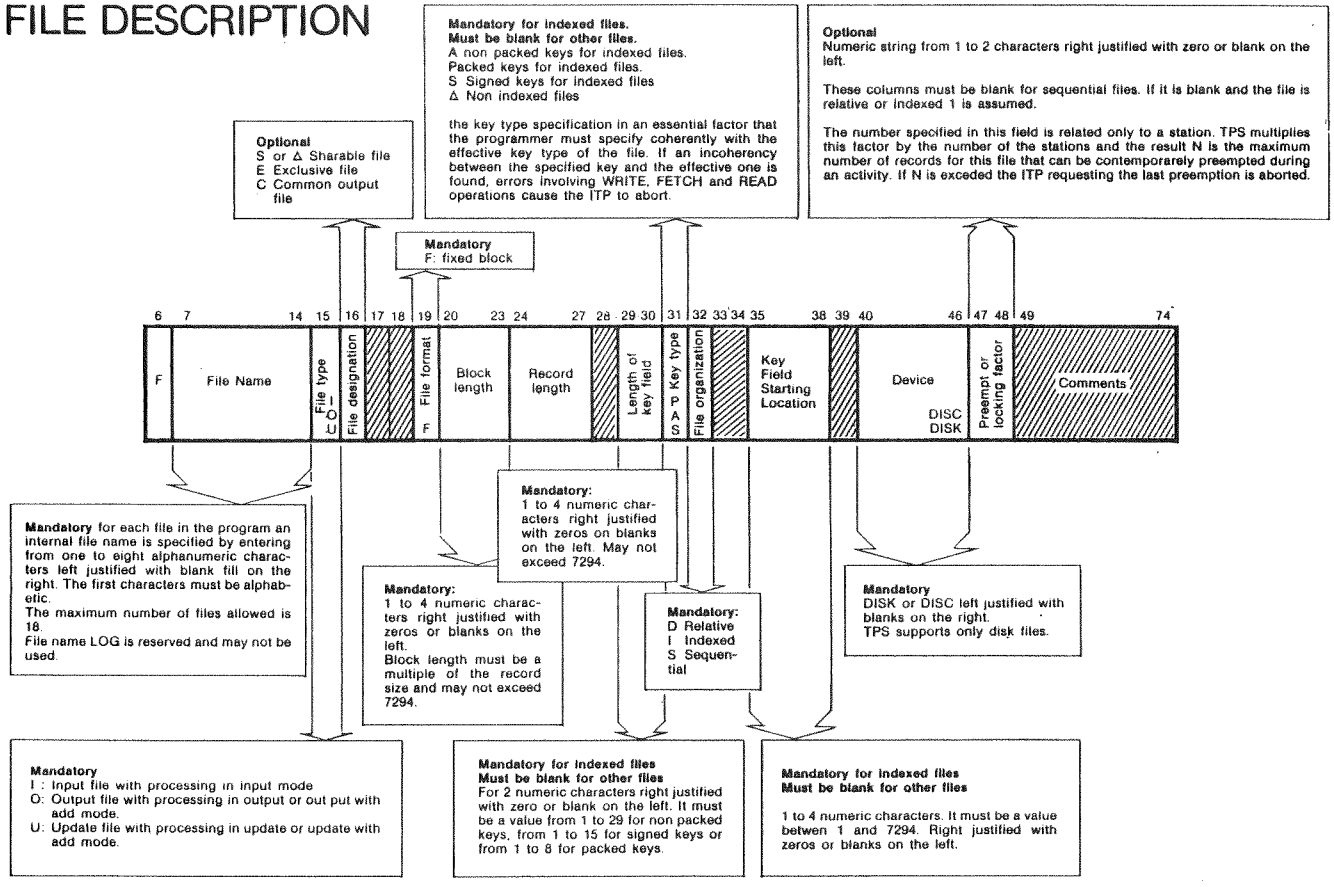


Fig. 3: Un esempio di "reference page": tutta la sintassi di uno statement in uno schema compatto.

6.2. Le pagine di riferimento

Le pagine di riferimento contengono invece tutte quelle informazioni di dettaglio per un certo uso del prodotto che è opportuno tenere concentrate. Esse potranno essere estraibili (stampate su cartoncino), non avranno struttura di modulo e non saranno, in generale, utilizzabili come "visuals " per proiezione (la densità di informazione sarà, in genere, molto alta). Per distinguerle da queste ultime, le pagine di riferimento non sono contornate dalla cornice standard.

7. STRUTTURA A MODULI E ATTIVITA' DI FORMAZIONE

Un manuale organizzato in moduli nel modo illustrato può essere direttamente utilizzato nell'attività di formazione sul prodotto da esso descritto. Basterà, infatti fotocopiarne tutte le pagine sinistre su trasparenti per lavagna luminosa per avere a disposizione, senza sostanziali costi aggiuntivi, un corso completo sul prodotto.

Le pagine destre serviranno, allora, sia come guida per l'istruttore che come testo di supporto per lo studente. Questi potrà completarle e personalizzarle, se necessario, con appunti presi durante la lezione del docente (°).

(°) La struttura a moduli, infatti, comporta in genere che una gran parte della pagina destra rimanga bianca.

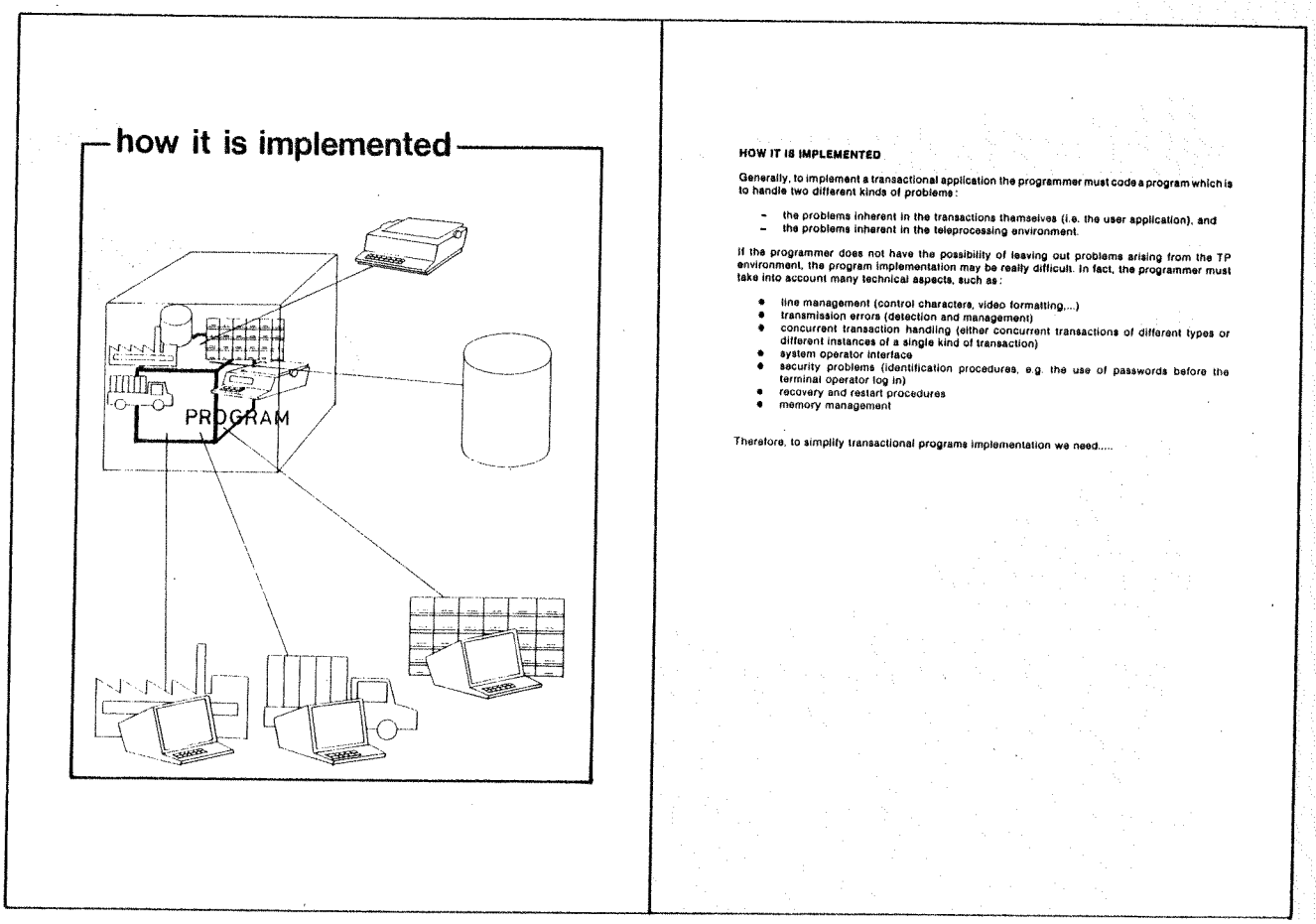


Fig. 4: Un esempio di modulo PSPN

8. CONCLUSIONI

Si è presentata una tecnica di organizzazione di manuali per l'utente che permette di riunire, in un unico prodotto:

- la documentazione introduttiva ("user guide")
- la documentazione di riferimento ("reference manual")
- la documentazione per la formazione (corsi)
- strumenti d'uso ("reference cards").

Tale tecnica si basa su:

- un insieme di linee guida sull'organizzazione degli argomenti (suddivisione per utenti e per uso, struttura gerarchica, concentrazione dell'informazione)
- uno standard di presentazione basato sulla struttura "a moduli PSPN"

e offre i seguenti vantaggi :

- facile accessibilità dell'informazione
- possibilità di valutarne facilmente la completezza
- possibilità di valutarne facilmente la gradualità e l'organicità
- leggibilità a diversi livelli di dettaglio
- contenuti costi di manutenzione
- identificazione degli sforzi volti alla documentazione e alla didattica.

9. RIFERIMENTI

1 TPS MANUAL (H.I.S.I.)

2 G.Gobbi, M.Maiocchi, AUTODOCUMENTAZIONE DEI PROGRAMMI, NOTE DI SOFTWARE 3

3 G.Armanini, R.Candela, S.Leva, R.Mari, S.Mazzocchi, LA REALIZZAZIONE DELLA DOCUMENTAZIONE PER UTENTI DEL LIVELLO 62, NOTE DI SOFTWARE 3

4 G.Degli Antoni, G.Torriani, DOCUMENTAZIONE E PRODUTTIVITA', XIV Convegno Internazionale di Automazione e Strumentazione, Milano, 23-24 novembre 1976 , e NOTE DI SOFTWARE 3.

NOTA

Le idee e le esperienze presentate dagli autori della presente nota nascono in realtà dalla collaborazione di molte persone. La tecnica di presentazione a moduli "PSPN" è stata messa a punto e lungamente sperimentata da varie persone presso l'Università di Milano, in particolare G. Degli Antoni, M. Maiocchi e numerosi laureandi dell'Istituto di Cibernetica.

Molte utili indicazioni per la costruzione del manuale di TPS sono state inoltre fornite da G. Degli Antoni, P. Lisci, S. Leva, E. Uggeri, B. Zonta.

VALUTAZIONE E MIGLIORAMENTO DELLE PRESTAZIONI DI STNAL

L. Becattini (°) - F. Faidutti (+)

1. INTRODUZIONE

STNAL (1,2) è un insieme di macroprototipi realizzati per facilitare la programmazione nell'assembler NAL del Livello 62.
Tali macroprototipi permettono di codificare direttamente le strutture di controllo tipiche della programmazione strutturata, proprie dei linguaggi di alto livello.

Ogni chiamata di un macroprototipo viene interpretata dal macroprocessor (3, cfr. anche 4), il quale la espande in opportune istruzioni elementari NAL.
Questa fase di macrogenerazione può influire pesantemente sui tempi di compilazione.

Lo scopo del lavoro i cui risultati sono sintetizzati nella presente nota è stato quello di :

- valutare l'appesantimento dei tempi di compilazione dovuto all'uso di STNAL
- introdurre le opportune modifiche ai macroprototipi per ridurre tali tempi
- introdurre alcuni miglioramenti nelle macro, come l'opzione ELSE IF nella struttura IF THEN ELSE, la ripetitività della struttura di Zahn EXECUTE , e opportuni controlli sulla corretta nidificazione delle macrocall.

(°) H.I.S.I., Pregnana Milanese
(+) Istituto di Cibernetica dell'Università di Milano e H.I.S.I., Pregnana Milanese.

2. VALUTAZIONE DELLE PRESTAZIONI DEL VECCHIO STNAL

I grafici in fig.1 riguardano procedure campione, divise in gruppi (uno per struttura). Ogni gruppo è costituito da cinque procedure, consistenti esclusivamente di macrocalls; il numero di strutture complete richiamate da ogni procedura cresce da un minimo di dieci a un massimo di cinquanta, con intervalli di dieci. Il tempo, in ordinata, è misurato in secondi, su una scala semi logaritmica.

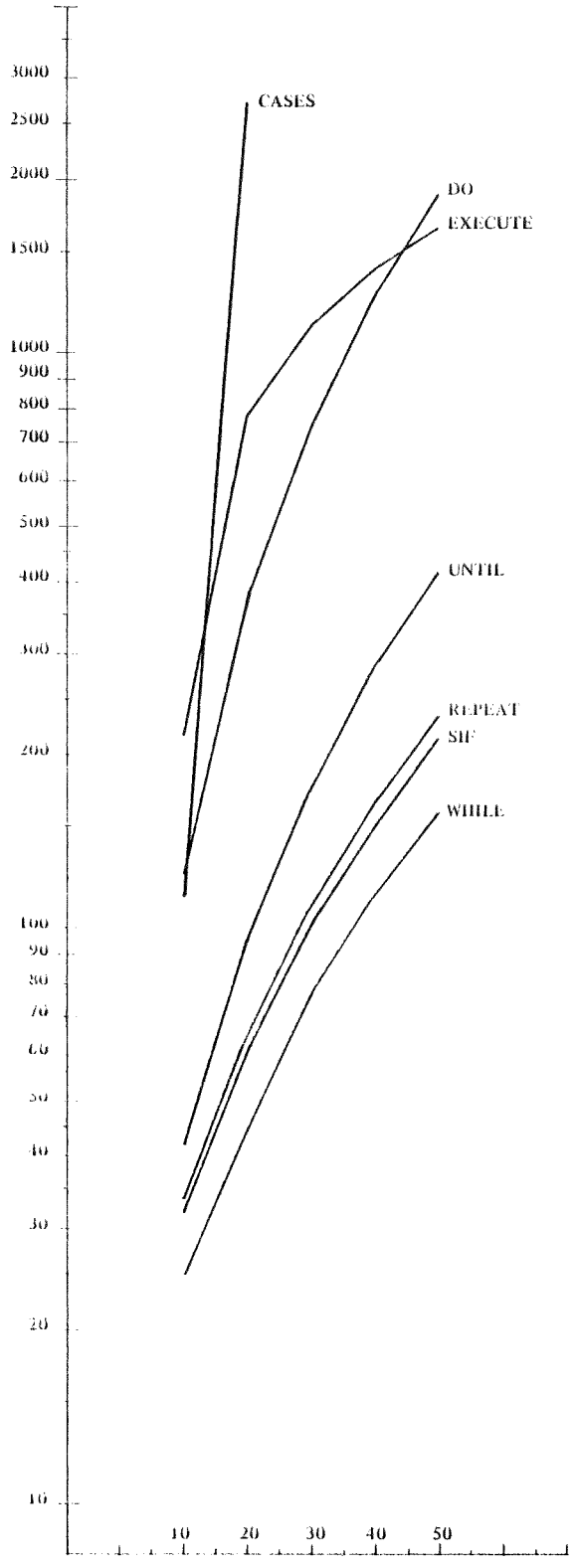
Da alcune misure fatte su procedure reali, risulta che il tempo di macrogenerazione può arrivare al 30% della intera compilazione: ad esempio, per una procedura composta da 3798 righe di codice NAL e 89 chiamate di macroprototipi di STNAL, il tempo di compilazione è stato di 34' 02". Di questi , 9' 45" sono stati utilizzati dal macroprocessor per espandere le macrocall.

3. ANALISI DEL VECCHIO STNAL

Da un esame del vecchio STNAL risulta chiaro che la causa principale dei notevoli tempi di compilazione indicati nel paragrafo precedente risiede :

- Nel fatto che la ricerca degli errori viene effettuata al livello più alto possibile, cioè a macro-generation-time, allo scopo di fornire all'utente una diagnostica molto selettiva e dettagliata. Un gran numero di operazioni di macroprocessor e di chiamate innestate è dovuto ai numerosi controlli sintattici presenti nei macroprototipi. Essi riguardano la consistenza dei parametri di chiamata e la loro correttezza formale. Esi-

FIGURA 1



stono inoltre altri tipi di controlli sulla corretta apertura e chiusura delle strutture.
Per ogni errore riscontrato, viene generato dai macroprototipi un messaggio che specifica in dettaglio il tipo di errore commesso dall'utente.(Il numero complessivo di questi messaggi è 80, con una occupazione media di 50 caratteri ognuno).

- Nell'elevato numero di chiamate innestate.
  La tendenza nel disegno del vecchio STNAL è stata, infatti, quella di incapsulare in macro di servizio (invisibili all'utente) molte funzioni logiche (controlli sui parametri di chiamata, accessi agli stack di globali, ecc.).
  Queste scelte erano motivate da considerazioni di segmentazione e manutenibilità dei macroprototipi.
  Tali macroprototipi di servizio vengono richiamati all'interno delle macro visibili all'utente, il che comporta operazioni di lettura dalle librerie di macro, su disco,e di interfaccia con il macroprototipo chiamante, operazioni che innalzano i tempi di sviluppo del macroprototipo.
- Di conseguenza, nella elevata dimensione (in bytes) dei macroprototipi.
  Ogni volta che il macroprocessor incontra una macrocall (nel sorgente o all'interno di un macroprototipo), provvede a trasferire la corrispondente macrodefinizione dalla libreria in cui essa risiede alla memoria principale.
  Tale trasferimento viene eseguito a blocchi di 496 bytes per volta, fino a esaurimento della macrodefinizione, cioè fino a che viene incontrato lo statement $END.
  I macroprototipi vengono accodati nel 'work stack', il quale deve essere ristrutturato ogni volta che si presente una situazione di 'overflow'. Tale situazione è determinata dalla lunghezza dei macroprototipi, ed è chiaro che le operazioni di ristrutturazione dello stack sono tanto più frequenti quanto più essi sono lunghi e "sparpagliati" (nell'ordine in cui sono incontrati e utilizzati).
  Sono proprio questi tempi di caricamento e di ristrutturazione che influiscono in maniera determinante sui tempi di compilazione.

La lunghezza dei macroprototipi è un effetto delle caratteristiche esposte nei due punti precedenti.

4. IL NUOVO STNAL

Tenendo conto di quanto esposto nel paragrafo precedente, si è pensato di ridisegnare il prodotto, con l'obiettivo di migliorare decisamente i tempi di compilazione e, inoltre, di

- introdurre controlli sulla corretta nidificazione delle strutture che impediscano un uso scorretto del linguaggio. La correttezza di un programma strutturato deve essere controllata non solo sulla sintassi della singola frase o della singola struttura, ma anche a livello delle relazioni di una struttura con tutte le altre, dello stesso tipo o no.
- introdurre alcune opzioni come la ELSE IF e il parametro LOOP che renda (opzionalmente) ripetitiva la struttura EXECUTE.
- mantenere e/o migliorare le caratteristiche del codice espanso per quanto riguarda occupazione di memoria e tempi di esecuzione.

Nel rifacimento, ci si è attenuti alle seguenti direttive:

- mantenere inalterata la visibilità utente e, fin dove possibile, il codice espanso.
- eliminare tutti i messaggi diagnostici e ridurre notevolmente i messaggi commento.
  L'appesantimento delle dimensioni dei macroprototipi del vecchio STNAL era dovuto, oltre che ai messaggi diagnostici descritti al paragrafo 3, ai messaggi espansi sotto forma di commento allo scopo di racchiudere il codice espanso dalla singola macro, per una più facile identificazione. (Questi messaggi erano due per ogni macroprototipo, con una occupazione media di 80 caratteri complessivi).
  Nel nuovo STNAL esistono solo due messaggi di errore standard: 'WRONG MACRO CALL' per gli errori formali, e 'WRONG MACRO NESTING' per gli errori di strutturazione.
  Essi figurano una volta sola nel corpo di ogni macroprototipo. I messaggi commento sono ridotti ad un'unica riga di identificazione delle strutture, che occupa mediamente 10 caratteri.
  Queste scelte hanno portato a una notevole diminuzione dell'occupazione di memoria (cfr.la tabella 1).
- ridurre pesantemente il numero dei macroprototipi di servizio.
  Essi sono stati portati da 17 (tanti erano nel vecchio STNAL) a 2. Ciò ha diminuito i tempi e il numero di operazioni di I/O descritti nel paragrafo 3, ed ha ottimizzato ulteriormente l'occupazione di memoria.

I risultati ottenuti dopo il rifacimento del prodotto sono visibili nella tabella 1 per l'occupazione di memoria e nei grafici di fig.2 e 3 per i tempi di compilazione. Nel grafico di fig. 2 sono riportate le curve ottenute compilando le procedure di cui al par.2 con il vecchio STNAL (linea continua) e con il nuovo (linea tratteggiata). I tempi sono riportati in ordinata su scala logaritmica. Il grafico di fig.3 rappresenta la percentuale di guadagno ottenuta compilando le procedure campione con il nuovo STNAL piuttosto che con il vecchio.

5. CONCLUSIONI

Le scelte fatte sono state dettate da una differente filosofia di diagnostica rispetto al vecchio STNAL: la ricca diagnostica di quest'ultimo proteggeva l'utente scorretto e facilitava la fase di debugging dei programmi; mentre però pochi usufruivano di queste caratteristiche, tutti pagavano gli eccessivi tempi di compilazione ad esse dovuti.

E' stata eseguita una serie di prove per verificare quantitativamente i risultati ottenuti. Una tra le più significative è consistita nella compilazione di due procedure di lunghezza media. Tale compilazione è stata eseguita utilizzando prima il vecchio STNAL e poi il nuovo. I risultati sono riportati nella tabella 2. Da questa si vede che, mediamente, il tempo di espansione si è ridotto del 35%. Per quel che riguarda il rapporto tra il tempo totale di compilazione e il solo tempo di macrogene-

Tabella n. 1

TABELLA COMPARATIVA DELLE DIMENSIONI DEI MACROPROTOTIPI

Macro visibili	vecchio STNAL (bytes)	nuovo STNAL (bytes)	-%	macro di servizio	vecchio STNAL (bytes)	nuovo STNAL (bytes)
SIF	1.429	547	60	QSCAN30P	-	1.208
ELSE	717	703	2	QSCAN60P	-	503
ENDIF	621	466	25	CARNUM	613	-
WHILE	1.524	499	67	CONC	27	-
UNTIL	1.441	533	62	ISNUM	46	-
DO	3.495	2.268	35	ISNUMBER	225	-
ENDDO	2.019	568	72	ISREG	820	-
REPEAT	558	317	43	MSTACK	253	-
EXIT	1.589	534	66	M1	14	-
ENDREP	853	340	60	POP	160	-
EXECUTE	1.442	613	57	PUSH	368	-
EVENT	1.977	858	57	QASCAN	299	-
THEN	873	478	45	QASEP	125	-
WHEN	1.723	643	62	RESEARCH	223	-
ENDEXEC	683	271	60	SCANGLB	32	-
STRTCASE	961	422	56	SEARCH	117	-
CASES	1.238	1.021	18	STSORT	711	-
ENDCASE	1.458	945	35	TOP	283	-
LISTING	....	104				
STNAL	..	141				

razione, si passa dal 16% al 12% per la procedura INITREP, e dal 23% al 17% per OKORNOK.

Altre prove del tipo suddetto sono state eseguite dando luogo a una ulteriore scoperta molto significativa: in molte procedure che prima venivano compilate correttamente, sono stati riscontrati errori del tipo 'WRONG MACRO NESTING' dovuti a un uso scorretto delle strutture.

Tabella 2

Procedura	tempo totale (compilaz. + macroespans).	tempo di macroespansione	
INITREP	8' 26"	1' 21"	Vecchio STNAL
OKORNOK	4' 14"	0' 57"	
INITREP	6' 41"	0' 52"	Nuovo STNAL
OKORNOK	3' 15"	0' 35"	

Ciò ha provocato l'espansione di molto codice in meno da parte del nuovo STNAL, con conseguente diminuzione nei tempi di compilazione. I risultati sono mostrati nella tabella 3.
Tali prove hanno evidenziato che l'assenza di alcuni controlli di tipo "contestuale" (nidificazione delle strutture) nel vecchio STNAL permettevano usi non completamente coerenti con i principi della programmazione strutturata (un ingresso/una uscita). I punti critici sono localizzati prevalentemente sull'uso della struttura REPEAT-EXIT: il macroverbo EXIT, concepito per realizzare l'uscita da un ciclo incondizionato, al suo proprio livello, (fig.4), è stato invece utilizzato, nelle procedure esaminate, anche per realizzare uscite da strutture di controllo qualsiasi, nidificate nel ciclo originario (ad es., fig.5).

FIGURA 2

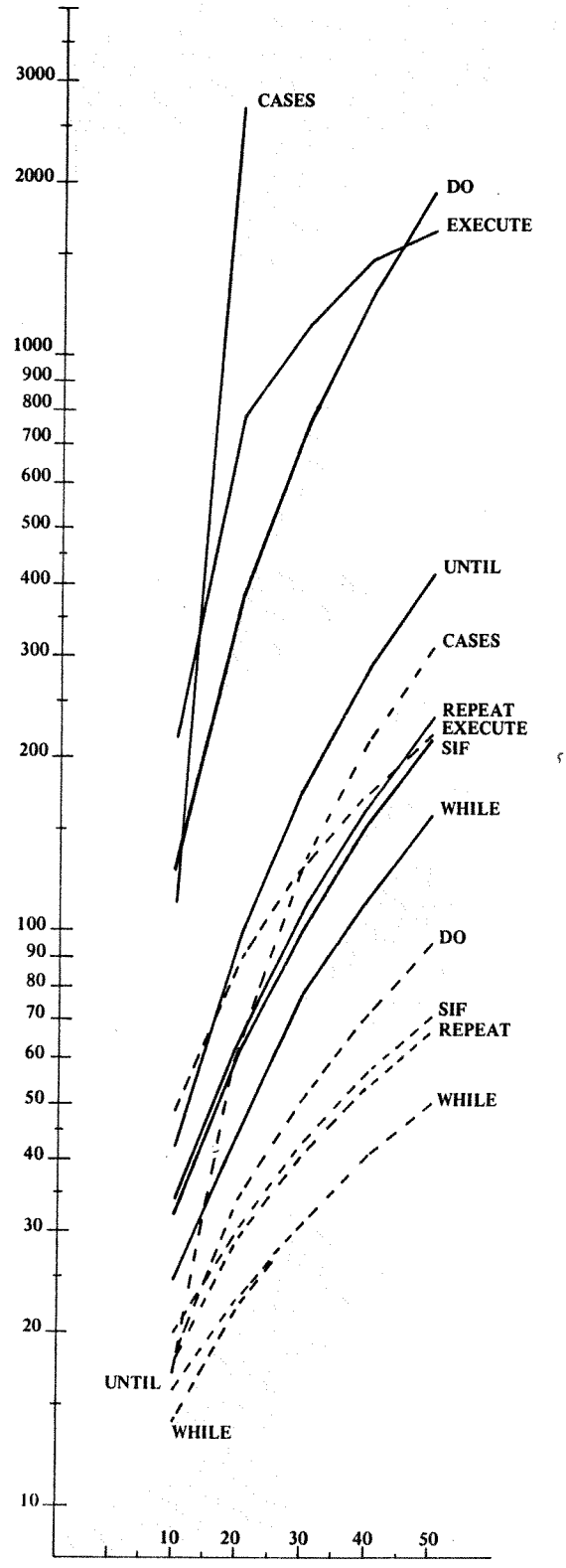


FIGURA 3 - %GUADAGNI

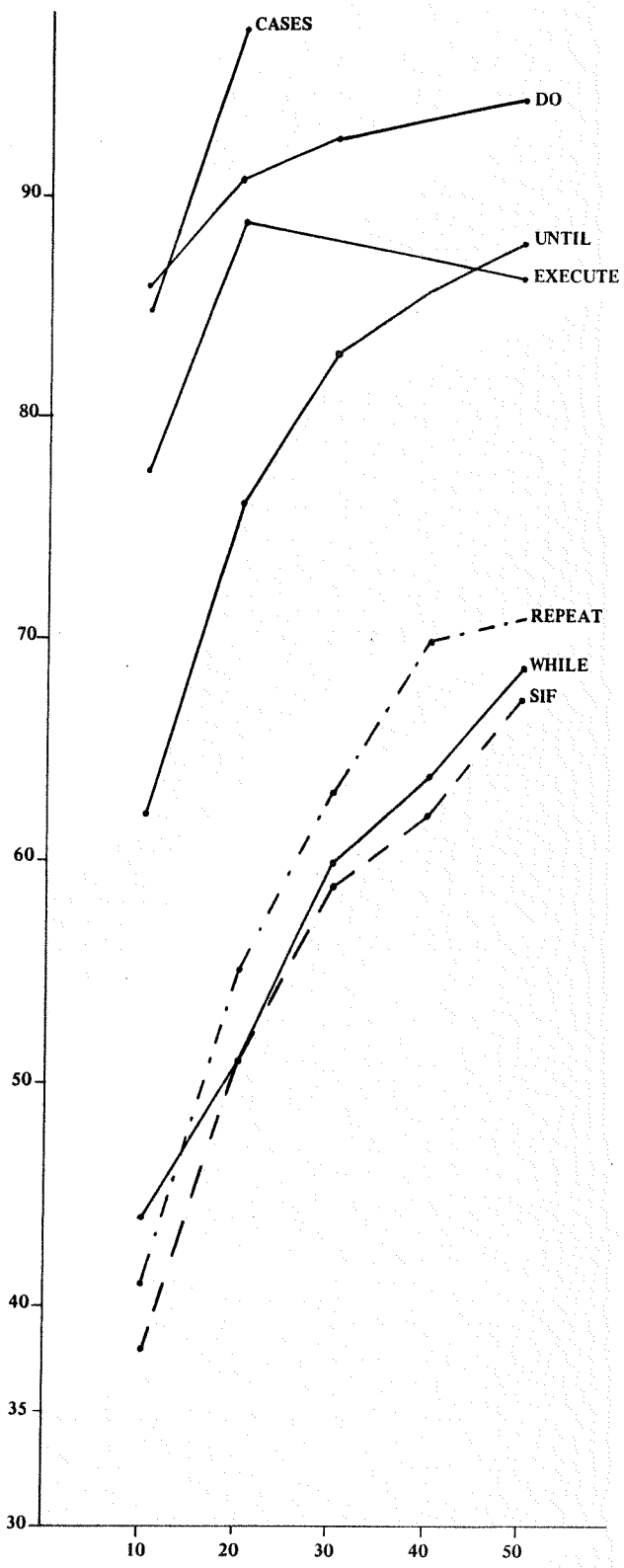


Tabella 3

Procedura	tempo totale (compil. e macroespansione)	tempo di macroespans.	
SCGO	28'01"	10'20"	vecchio STNAL
	14'43"	2'08"	nuovo STNAL
STDSKTSL	16' 0"	4'33"	vecchio STNAL
	7'23"	1'22"	nuovo STNAL

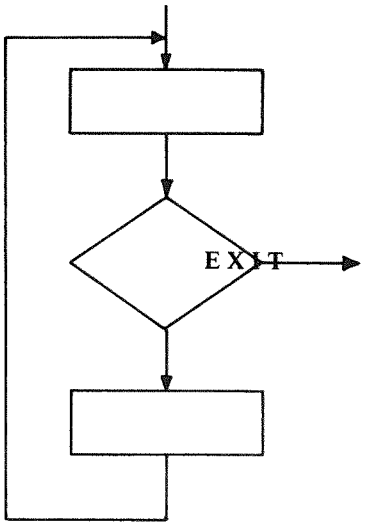


fig. 4

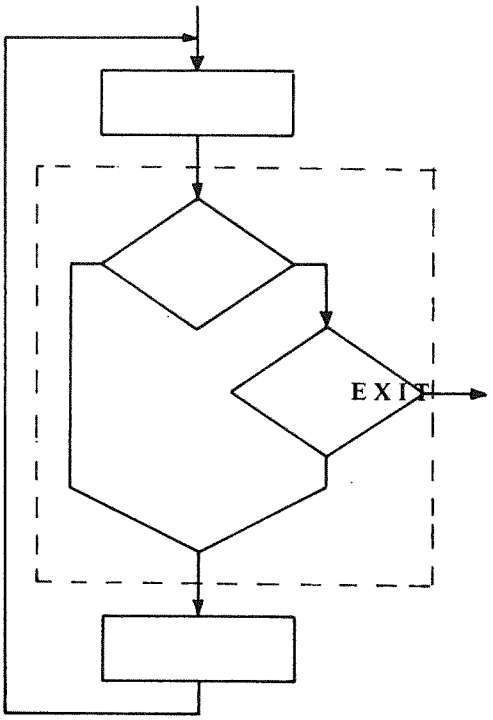


fig. 5

6. RIFERIMENTI

1. STNAL: MACROLIBRARY FOR STRUCTURED PROGRAMMING IN NAL - Functional Specifications (MN 0200.03), documento interno H.I.S.I., Pregnana Milanese.

2. S.Farnedi, M.Maiocchi, C.Poggiali, R.Polillo, STNAL: UN INSIEME DI MACRO PER LA PROGRAMMAZIONE STRUTTURATA IN NAL, NOTE DI SOFTWARE n.1.

3. MACROPROCESSOR Functional Specifications (MN 0157.48), documento interno H.I.S.I., Pregnana Milanese.

4. V.Cajani, R.Rousseau, MACROPROCESSORS : GENERAL CONCEPTS, NOTE DI SOFTWARE n.2.

UN'OPPORTUNA STRUTTURA DI DATI, PUO' SEMPLIFICARE LE STRUTTURE DI CONTROLLO?

L. Antonelli - H.I.S.I. - Pregnana Milanese

1. INTRODUZIONE

Il problema si pone in fase di impostazione di un prodotto, e la risposta comporta delle scelte che ne caratterizzeranno per sempre la struttura.
Nell'esposizione che segue si intende illustrare l'ambiente in cui si pone il quesito, ossia l'analisi di un prodotto, e la possibilità di una risposta affermativa.

2. ANALISI DI UN PRODOTTO

Il lavoro di analisi di un prodotto da sviluppare può essere suddiviso in varie fasi.

2.1. Analisi delle funzioni fondamentali

Prima di tutto si procede alla definizione delle funzioni fondamentali da svolgere e per quanto possibile di quelle ausiliarie (quali ad esempio il trattamento di condizioni anomale o fasi fuori ciclo).
Lo studio attento delle funzioni fondamentali deve condurre alla stesura di un brevissimo elenco di operazioni complesse, che in seguito rimanderanno ad altre operazioni che saranno collocate a livelli gerarchici inferiori.
La fase successiva è quella di individuare il concatenamento di queste operazioni complesse, ossia il loro imbrigliamento in strutture di controllo (sequenze o iterazioni) costituenti un primo flusso.

2.2. Analisi delle funzioni ausiliarie

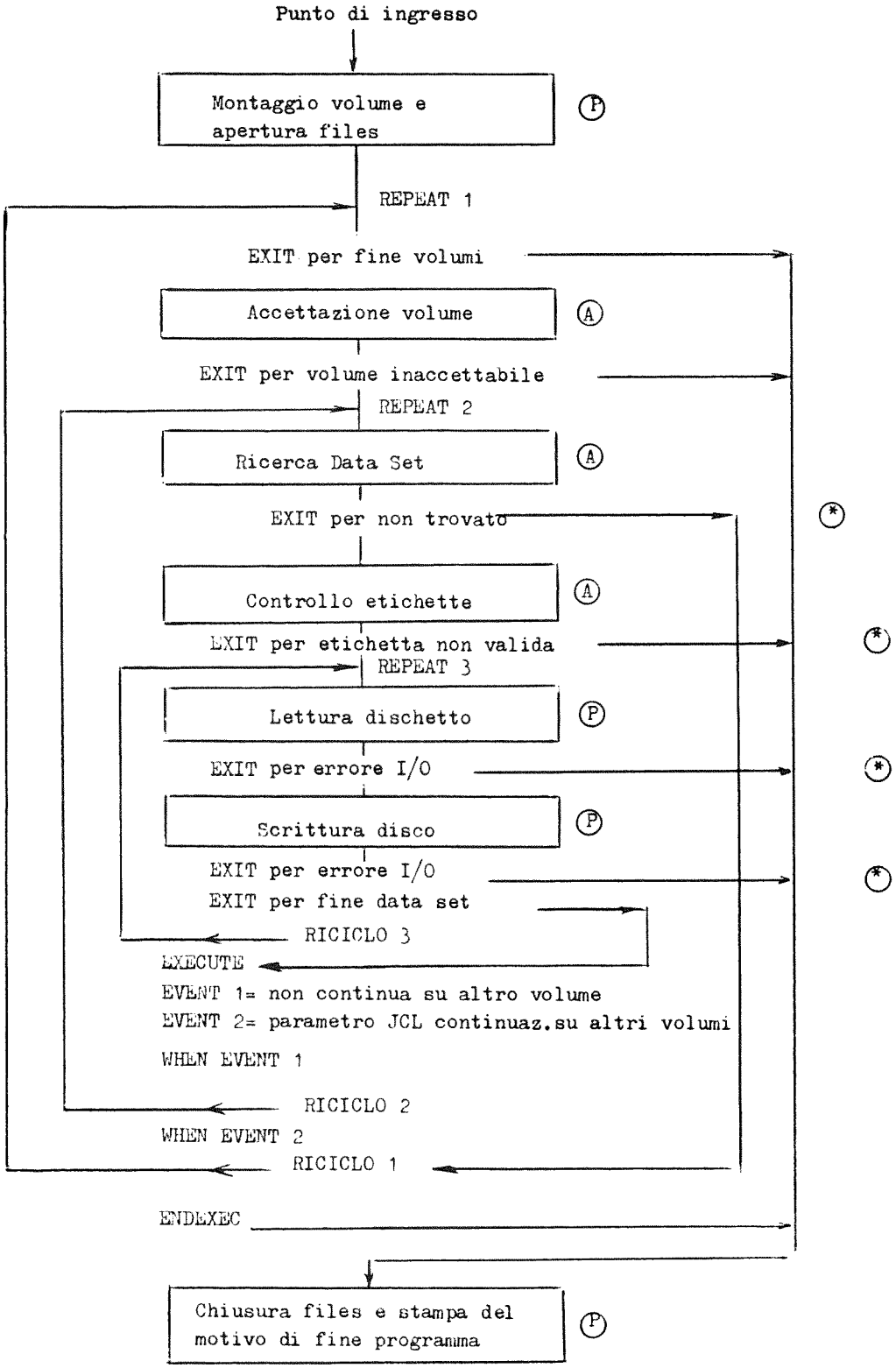
La terza fase consiste nello studio delle funzioni ausiliarie, e quindi la definizione di altre operazioni la cui complessità dipende dalle dimensioni del prodotto e quindi dalla quantità di livelli gerarchici che sottostanno.
Il problema che ora si pone è quello di inserire e concatenare le operazioni relative alle funzioni ausiliarie nel flusso delle operazioni precedentemente definite e collegate.
E' probabile che a questo punto le strutture di controllo precedentemente scelte non riescano ad inglobare le nuove operazioni, o che comunque la linearità del primo flusso sia pesantemente turbata.

3. AZIONI POSSIBILI

Le soluzioni sono diverse.
Una è quella di individuare una nuova intelaiatura di strutture di controllo che colleghino in modo semplice e lineare le operazioni sia fondamentali che ausiliarie, il che non è sempre possibile.
Una seconda è quella di rinunciare alla linearità e quindi alla leggibilità e manutenibilità del programma.
Una terza è quella di definire una struttura di dati, che non necessariamente abbia interfaccia verso l'esterno del programma, e che costituisca un deposito organizzato di informazioni che si propagano tra le varie sotto - parti; questo può comportare un lieve appesantimento delle strutture precedenti, ma non complicazione nè aggravi di codice che debba ricostruire informazioni andate perse durante il flusso del programma.
Si ottiene così una interfaccia generalizzata, costituita da più elementi, di cui ognuno è chiaramente definito, organizzato fisicamente vicino agli altri, e con i quali costituisce un'unica "password" per passare da una struttura ad un'altra, o risalire nei livelli di nidificazione in modo ordinato.

4. CONCLUSIONE

Risultato pratico della terza soluzione è che la leggibilità e la linearità del flusso sono mantenute, con conseguenti vantaggi nel debugging e nella manutenzione del prodotto.
L'appesantimento della zona dati dipende dalla quantità e dimensione degli elementi dell'interfaccia generalizzata, che dovrebbe essere contenuta entro una spazio modesto rispetto al



(P) Funzioni principali

(A) Funzioni ausiliarie

FIGURA 1

la dimensione del prodotto, per evitare che la farraginosità delle strutture di controllo si trasferisca sulle strutture dei dati.

5. UN ESEMPIO

Quale esemplificazione si presenta il caso di un programma di utilità di copiatura dati da dischetto a disco, scritto in linguaggio NAL delle dimensioni di circa 4000 byte di codice.

5.1. I dati del problema

- Parametri di definizione modalità di copiatura da utente (via J.C.L.)
- Dati organizzati in gruppi di records detti "data set"
- I data sets possono risiedere in più volumi

5.2. Prima analisi delle funzioni

Si individuano tre funzioni fondamentali

- montaggio volumi e apertura files
- nucleo operativo di lettura dischetto e scrittura dati su disco
- chiusura files.

Le funzioni ausiliarie sono

- controllo fine volumi e validità etichetta-volume
- ricerca data set e controllo etichetta
- controllo errori operazioni input/output
- messaggi finali.

Le corrispondenti operazioni complesse vengono concatenate in tre livelli di nidificazione come mostra la figura 1, in cui si nota che:

- le EXIT contrassegnate da ⊛ male si conciliano con le tre REPEAT perchè scavalcano più livelli di strutture
- la EXECUTE finale ad un esame più dettagliato, in fase di analisi di funzioni ausiliarie, non consente di risolvere completamente i problemi
- non è facile risalire alla causa di fine programma per dare opportuno messaggio.

5.3. Seconda analisi

L'introduzione di una struttura di dati e una riorganizzazione del flusso precedentemente definito è il risultato della rianalisi del prodotto.

La struttura dati è costituita da un byte (chiamato EXITFLAG) i cui bit hanno ciascuno un significato preciso come indicato nella figura sotto riportata.

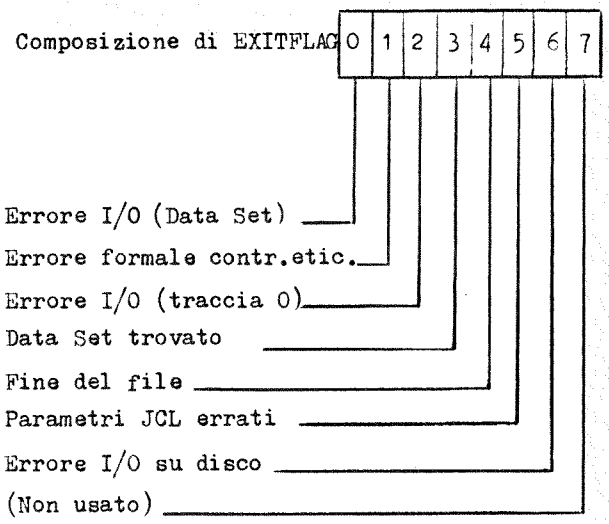


FIGURA 2

La struttura di controllo assume la forma della figura 3, in cui si nota che

- la struttura EXECUTE scompare
- quasi tutte le EXIT sono dei test su EXITFLAG
- anche quelle condizioni che porterebbero ad una uscita immediata dal flusso (segnate con ⊛ in fig. 1) rimandano il controllo solo all'anello successivo da cui eventualmente rimbalzano ulteriormente
- alla CLOSEFILE rimane traccia precisa per la scelta del messaggio finale.

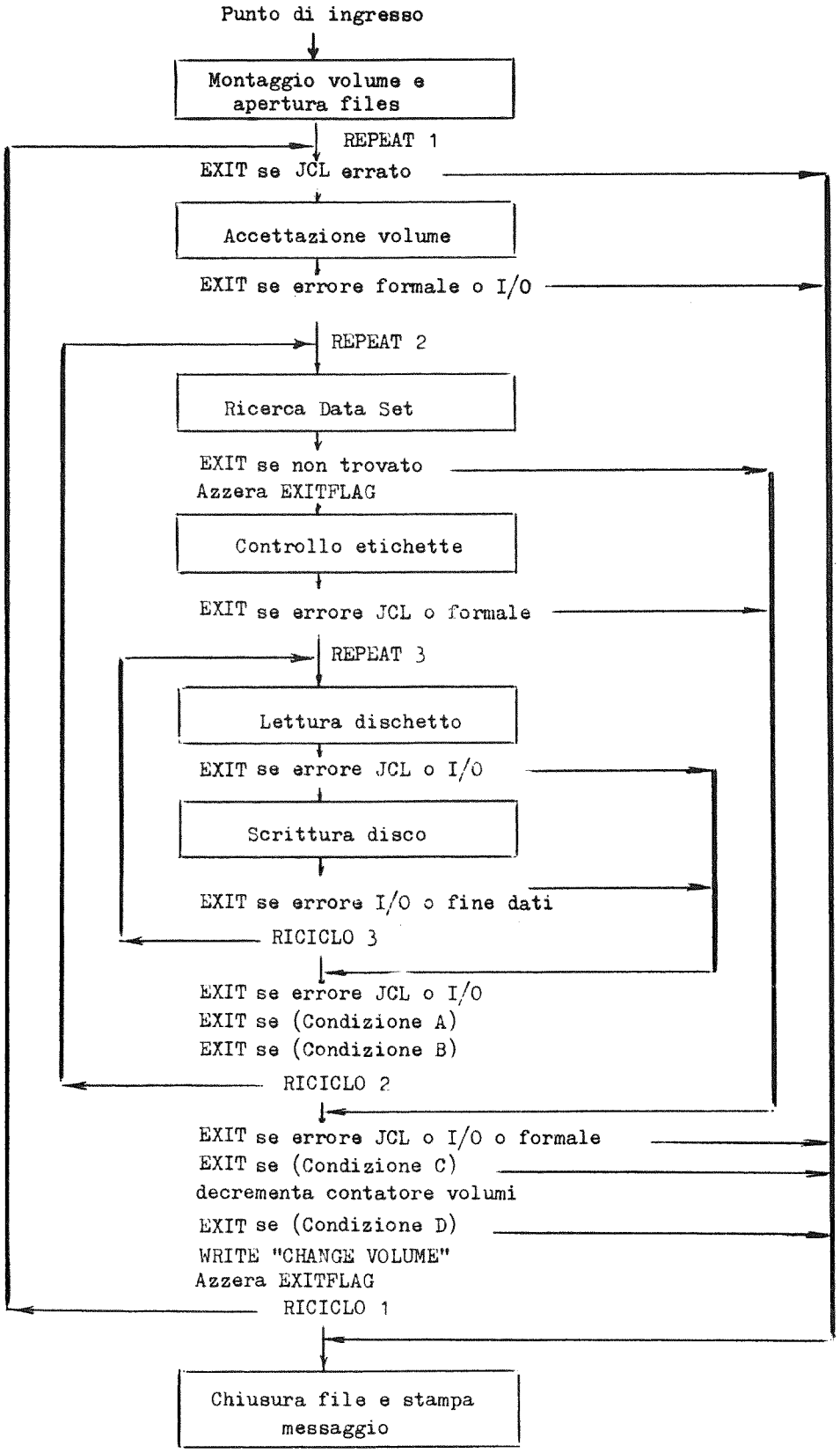

Punto di ingresso
Montaggio volume e apertura files
REPEAT 1
EXIT se JCL errato
Accettazione volume
EXIT se errore formale o I/O
REPEAT 2
Ricerca Data Set
EXIT se non trovato
Azzera EXITFLAG
Controllo etichette
EXIT se errore JCL o formale
REPEAT 3
Lettura dischetto
EXIT se errore JCL o I/O
Scrittura disco
EXIT se errore I/O o fine dati
RICICLO 3
EXIT se errore JCL o I/O
EXIT se (Condizione A)
EXIT se (Condizione B)
RICICLO 2
EXIT se errore JCL o I/O o formale
EXIT se (Condizione C)
decrementa contatore volumi
EXIT se (Condizione D)
WRITE "CHANGE VOLUME"
Azzera EXITFLAG
RICICLO 1
Chiusura file e stampa messaggio

FIGURA 3

TMSS: UNO STRUMENTO PER LA GESTIONE AUTOMATICA DI CHECK-LISTS DINAMICHE NELLA QUALIFICAZIONE INDUSTRIALE DEL SOFTWARE

A. Cazzio[°] - C. Cucciati[+]

Riassunto. Il processo di qualificazione del software in ambienti di produzione industriale pone problemi che non sono solamente tecnologici, ma spesso in grande misura anche organizzativi e gestionali.

Il presente articolo si propone innanzi tutto di individuare ed analizzare tali categorie di problemi, collocandoli in un contesto ben definito: quello di un ambiente dove si progetta e si costruisce il software di sistema per un elaboratore di medie dimensioni.

Le esperienze da cui trae origine l'articolo sono infatti quelle realizzate presso la Sezione Ingegneria della HISI, a Pregnana Milanese.

Attraverso l'analisi di alcuni problemi tipici (strategie, costi, controllo di avanzamento, valutazioni di copertura, ecc.) l'articolo cerca di precisare i requisiti fondamentali di strumenti o di procedure di gestione per le attività di qualificazione.

Vengono quindi descritte in dettaglio le funzionalità, la struttura e le modalità d'uso di uno strumento specifico, TMSS (Test Management Support System), realizzato a Pregnana, e capace di fornire una parziale soluzione ai problemi esposti.

Del TMSS vengono inoltre descritte le possibilità di integrazione con altri strumenti automatici per la gestione della qualificazione.

(+) Honeywell I.S.I.
Pregnana Milanese
(o) Università di Milano-Istituto di Cibernetica, e H.I.S.I.

1. INTRODUZIONE

I problemi della qualità del software, ed in particolare quelli relativi al testing, sono stati frequentemente affrontati negli ultimi tempi e notevoli contributi sono stati dati all'evoluzione dei criteri e delle tecniche di costruzione ed esecuzione dei test.

La tendenza generale in questi casi è stata ed è tuttora quella di porre l'accento soprattutto sui problemi tecnologici del testing dei programmi (copertura topologica, testing top-down, ecc), con l'obiettivo di fornire criteri e metodologie per migliorare l'efficacia e completezza "intrinseca" dei test..
Tuttavia esistono ambienti nei quali l'efficacia delle attività di testing dipende non solo dalla "bontà intrinseca" dei test, ma anche (e in misura non trascurabile) dal modo in cui l'ambiente stesso si organizza per valutarne e sfruttarne i risultati (problemi gestionali e di comunicazione).
Gli ambienti in cui tipicamente acquista significato questa affermazione sono quelli che possiamo definire "ambienti di produzione industriale" del software, in cui :

- le dimensioni dei prodotti realizzati sono notevoli
- l'impostazione dei processi di sviluppo è basata su criteri di suddivisione del lavoro
- il processo di sviluppo è "distribuito" nel tempo, nel senso che innanzi tutto è di lunga durata, e spesso è articolato in più release .

Un esempio di quanto appena detto può essere la realizzazione di un grosso progetto software, che a causa delle sue dimensioni viene suddiviso in aree funzionali, da svilupparsi separatamente e in parallelo.

Ogni area funzionale vede l'impegno contemporaneo di più persone, ciascuna delle quali realizza uno o più componenti specifici. I componenti realizzati vengono poi riuniti per dare corpo alle aree funzionali, e queste ultime vengono a loro volta raccolte a formare il sistema completo.

Dal momento in cui un singolo componente comincia ad esistere, fino al momento in cui l'intero sistema viene rilasciato, si assiste ad una continua preparazione ed esecuzione di test di generalità crescente :

- i test sui componenti singoli
- i test su insiemi di componenti (che chiameremo "complex")
- i test sull'intero sistema.

In un caso come quello descritto, mentre è conveniente che singoli progettisti o gruppi di sviluppo eseguano per conto loro tutte le attività di testing sui propri componenti, è tuttavia necessario che siano assicurati i mezzi per recuperare in ogni momento una visione di insieme su "che cosa deve essere fatto e come stanno andando le cose".

2. PROBLEMI NELLA QUALIFICAZIONE INDUSTRIALE DEL SOFTWARE

Il primo problema che possiamo individuare deriva dal fatto che "esiste un legame preciso tra la qualità di un singolo componente e la qualità dell'intero sistema". Questo significa ad esempio che le informazioni sul grado di copertura (funzionale, topologica, ecc.) raggiunto su di un componente e le caratteristiche dei suoi test devono essere messe in relazione non solo al componente, ma anche all'intero sistema.

Un altro problema è legato al fatto che "il numero e le caratteristiche dei test esistenti a un certo istante (test di componente, test di complex, test di sistema, ecc.) non sempre garantiscono un'equilibrata copertura del sistema complessivo".
Ciò è dovuto essenzialmente al fatto che i primi test nascono quando nascono i singoli componenti, e la loro costruzione è curata dalle stesse persone che li hanno realizzati. Altri test nascono poi, per provare "complex" di componenti, spesso ad opera di altre persone, che possono operare con criteri autonomi, con il conseguente rischio di squilibri di copertura o duplicazioni di sforzi.
Infine, vengono preparati altri test (test di sistema, ecc.) da altre persone ancora.

Un tale modo di procedere ha una sua precisa ragione: gli obiettivi da perseguire ai vari livelli del testing sono diversi, e richiedono l'adozione di criteri differenti.

E' necessario quindi recuperare, preferibilmente con uno strumento automatico, la misura precisa della copertura realizzata sul sistema nel suo complesso, sui vari complex funzionali e sui singoli componenti. Questa copertura dovrà essere espressa,ai vari livelli, con una misura omogenea, o comunque con misure confrontabili.
Ad esempio,la copertura di un componente può essere espressa in "porzioni di programma eseguite", o in "funzionalità elementari" provate, ecc. Quella del sistema può invece essere espressa in termini di macrofunzionalità, di interi complex esercitati, ecc. In questi casi, occorre quindi fare in modo che la copertura dei componenti possa essere poi accumulata e tradotta anche in termini di copertura del sistema.

Risolto il problema della misura di copertura, l'esigenza che si pone è quella di minimizzare il numero di test necessari. Solitamente la situazione si rivela caratterizzata dal fatto che:

- ogni test prova contemporaneamente più parti del sistema, e non solo un componente o un complex
- nello stesso tempo ogni parte del sistema, per essere completamente provata, necessita di più test diversi.

Poichè il processo di sviluppo può portare alla produzione di un numero enorme di test , spesso ridondanti tra loro e comunque variamente ricoprentisi, e poichè per varie ragioni ogni test deve essere ripetuto più volte (in varie condizioni, ecc.), è necessario poter estrarre, a seconda delle necessità, insiemi minimali di test

che consentano di raggiungere gli obiettivi voluti. In mancanza di tale possibilità l'unica soluzione consiste nell'usare sempre tutti i test esistenti, e le conseguenze in termini di costi (uomini, macchine e tempo) possono divenire anche molto pesanti. La selezione dei test da usare rappresenta uno degli aspetti di quella che viene comunemente definita la "strategia di qualificazione". Altri aspetti saranno indicati più oltre.

Un terzo problema si pone in relazione alla necessità di "sincronizzare le attività di costruzione con le attività di qualificazione".
In particolare, le difficoltà si pongono quando, ultimato lo sviluppo di componenti, si comincia a integrarli tra loro per dare vita al sistema. L'integrazione è una operazione che procede per gradi, quasi in modo continuo, e che ad ogni passo richiede una precisa opera di qualificazione; vale a dire che occorre continuare a ripetere test.
Il punto fondamentale è che le operazioni di qualificazione devono tenere rigorosamente lo stesso passo delle operazioni di integrazione (e,più in generale, di costruzione). Pertanto occorre sapere e descrivere prima tutto ciò che deve essere fatto (pianificazione) e controllare poi, passo passo, ciò che in realtà si fa (controllo di avanzamento).

Ciò significa esprimere in modo preciso quali test devono essere eseguiti, quando e in che condizioni particolari,nonchè verificare ai momenti stabiliti che cosa è stato fatto e con quali risultati.
Quando si dice "condizioni particolari" di test, ci si può riferire a molte cose; trattandosi di software di sistema, ad esempio, il problema sarà prevalentemente quello di prevedere tutte le possibili configurazioni sia software che hardware.

Un ultimo problema è legato alla "necessità di valutare l'attività di qualificazione in relazione ai suoi risultati e di valutare la qualità finale del sistema".
La valutazione dell'attività di qualificazione ha lo scopo di dosare l'allocazione ed eventuale riallocazione delle risorse (uomini, macchine, tempo, test).

La valutazione della qualità del sistema, ha invece lo scopo di consentire decisioni come :

- il sistema è a posto: può essere rilasciato
- il sistema non è a posto, non può essere rilasciato .

3. STRUMENTI DI SUPPORTO ALLA QUALIFICAZIONE INDUSTRIALE DEL SOFTWARE

Gli strumenti sui quali vogliamo puntare la nostra attenzione non sono gli strumenti destinati a migliorare la qualità dei test, ma piuttosto quelli destinati a facilitare la soluzione dei problemi di gestione e di valutazione ai quali abbiamo accennato.

Check-List

La check-list ha generalmente la forma di una tabella su cui compaiono, in testa alle righe, l'elenco degli "aspetti" (generalmente funzionalità) da provare, e, in testa alle colonne, l'elenco dei test esistenti o da costruire.
Un segno nel punto di intersezione tra una riga ed una colonna esprime il fatto che un certo test prova un certo aspetto.
La check list può essere utilizzata a qualunque livello, dal singolo componente all'intero sistema.
Si può prevedere l'impiego di check-list strutturate a più livelli, in cui la check-list di livello 'n' rappresenta una forma di sintesi di più check-list di livello 'n+1'. Diventa in tal caso possibile pensare a strumenti per effettuare la sintesi di check-list , e quindi procedere di livello in livello fino ad ottenere lo stato della check-list di sistema, a partire dalle check-list di tutti i componenti.

Piani

I piani, documenti che esprimono cosa deve essere provato in ciascun momento, costituiscono l'elemento guida per individuare, attraverso le check-list , quali test devono essere usati di volta in volta, ed inoltre, a consuntivo, su quali

parti del sistema si può procedere (perchè i test hanno dato esito positivo) e su quali invece occorre fermarsi.
Anche i piani possono essere sviluppati su più livelli, con differenti gradi di dettaglio.

Librerie di test

I test sono molti ; devono essere conservati, devono essere ripetibili, devono poter essere mantenuti ed utilizzati facilmente.
Tutto questo spinge ad utilizzare anche per i test apposite librerie, entro le quali essi vengono conservati, aggiornati, e da cui vengono prelevati ogni volta che occorre.
Non sempre tutto ciò di cui si compone un test può essere inserito in una"libreria" : certi test prevedono l'impiego di supporti particolari (schede, nastri, cassette, ecc.) oppure certe azioni da parte dell'operatore, e tutto ciò non si può inserire in libreria. In questi casi ugualmente gli elementi di un test debbono essere "registrati", almeno come documentazione scritta, catalogati ed archiviati in modo che ne sia garantita la reperibilità.

Strumenti per la gestione automatica di check-list

La gestione delle check-list ,e l'estrazione di informazioni riassuntive, può risultare faticosa specie se le check-lists si articolano in più livelli: strumenti specifici possono essere predisposti per l'automazione di tali operazioni, ed anche per realizzare il collegamento logico tra check-list e piani di qualificazione.

Strumenti per la gestione delle librerie di test

Quando i test sono stati inseriti in apposite librerie, rimane il problema di estrarre di volta in volta i test voluti e di corredarli di quanto è richiesto per la loro esecuzione. Inoltre, l'esecuzione stessa deve essere guidata e controllata.
Anche in questo caso è possibile individuare strumenti automatici che, in base a semplici indicazioni, provvedano a prelevare dalle librerie i test e quanto altro serve, inseriscano tutto nell'opportuno ambiente di test, e forniscano un aiuto, almeno parziale, a chi controlla la loro esecuzione.

Strumenti per la rilevazione automatica dei risultati dei test

Quando la dimensione delle attività di qualificazione è molto elevata (ad es., migliaia di esecuzioni di test), sono indispensabili strumenti che permettano di rilevare automaticamente i risultati dei test, e che producano rapporti riepilogativi.

Nel seguito di questo articolo descriveremo uno strumento particolare, TMSS, sviluppato a Pregnana Milanese, il cui scopo è quello di automatizzare la gestione delle check-list di sistema, a fronte di piani di test prestabiliti, e la produzione di reports riepilogativi dello stato di avanzamento della qualificazione.

5. TMSS - AMBITO DI IMPIEGO E NATURA DELLO STRUMENTO

TMSS è uno strumento che può essere impiegato nell'ambito delle attività di testing di singoli programmi o di interi sistemi software, con lo scopo di gestire check-list ed altre informazioni per :

- la pianificazione dei test
- il coordinamento ed il controllo di avanzamento dei test
- la valutazione in itinere del testing.

Lo strumento è costituito da un insieme opportuno di files, su cui vengono registrate tutte le informazioni previste (check-list, piani di test, risultati, ecc.), e da due programmi per la costruzione e aggiornamento dei files, e per la estrazione di tutti i report. Una macro JCL, avente lo scopo di semplificare l'operabilità del TMSS, completa il tutto. Il TMSS opera sulla base di comandi e di dati forniti in input tramite schede.

6. FUNZIONI REALIZZATE DAL TMSS

Le principali funzionalità di TMSS presenti alla data attuale possono essere così riassunte :

- gestione check-list statiche (stato di test /funzioni)
- reporting sullo stato di test /funzioni
- gestioni check-list dinamiche (pianificazione ed avanzamento testing)
- reporting su pianificazione ed avanzamento testing.

Altre funzionalità previste per lo strumento, ma non ancora implementate, potranno essere collocate nell'ambito di due problemi:

- determinazione di sottoinsiemi di test, soddisfacenti predefiniti criteri di copertura (es.: copertura minimale su insiemi di funzionalità)
- gestione di check-list strutturate su più livelli, con sintesi automatica delle check-list verso i livelli più alti (es.: PROGRAMMA ⟶ COMPLEX ⟶ SISTEMA)

Esaminiamo ora in dettaglio i quattro gruppi di funzionalità sopra indicati.

Gestione delle check-list statiche

Con il termine di "check-list statica" intendiamo il complesso di relazioni esistenti tra l'insieme delle funzionalità di un prodotto e l'insieme dei test che provano in tutto o in parte tali funzionalità. In particolare la check-list statica è costituita da un insieme di relazioni del tipo:

- il test A copre la funzionalità X
- il test A non copre la funzionalità Y
- la funzionalità Z è coperta dal test B
- la funzionalità K non è coperta da alcun test

ecc.

La check-list in questo caso è "statica" in quanto riflette una situazione che non dipende dal fatto che i test in questione vengano eseguiti oppure no.
Essa è idonea a rappresentare lo stato e le caratteristiche di copertura del "patrimonio di test " esistente in un certo momento, e consente quindi di :

- reperire quei test che provano una certa funzionalità specifica
- valutare la distribuzione dei test sulle varie funzionalità
- decidere la costruzione di nuovi test per riequilibrare le situazioni di copertura sbilanciata
- selezionare subset di test soddisfacenti a particolari criteri di copertura ecc. .

La check-list statica è soggetta ad aggiornamenti consistenti essenzialmente in:

- aggiunte, eliminazioni, variazioni di funzionalità
- aggiunte, eliminazioni, variazioni di test.

Il problema principale nell'esecuzione di questi aggiornamenti consiste nel ricreare ogni volta i giusti legami tra le funzionalità esistenti e i test che le coprono.

TMSS consente di eseguire queste operazioni, memorizzando sui propri files la check-list ed automatizzando la gestione dei legami tra test e funzionalità in base alle indicazioni fornite dall'utente; in particolare :

- Aggiunta di nuove funzionalità
  TMSS aggiunge semplicemente le nuove funzionalità a quelle già esistenti.

- Aggiunta di nuovi test
  TMSS aggiunge i nuovi test a quelli già esistenti, e per ciascuno di essi stabilisce un legame bidirezionale con le funzionalità coperte.

- Variazione delle funzionalità coperte da un test
  TMSS provvede a creare i nuovi legami tra il test e le funzionalità aggiunte, e ad eliminare i legami per le funzionalità da cancellare

- Cancellazione di un test
  TMSS provvede a cancellare anche tutti i legami con le funzionalità che dal test erano coperte

- Cancellazione di una funzionalità
  TMSS inibisce la cancellazione di una funzionalità, fintantochè risulta che esistano test che la provano.

In aggiunta a quanto descritto, TMSS arricchisce le informazioni contenute nella check-list con l'indicazione del complex di appartenenza di ogni test.
Infatti, anche se un test finisce per solle-

citare funzionalità appartenenti all'intero sistema, usualmente viene costruito per la qualificazione di un complex ben preciso. Ove la classificazione in base a "complex di appartenenza" non avesse motivo di esistere, TMSS consente di utilizzare quel tipo di informazione come un elemento di classificazione dei test in gruppi (in base al responsabile, o all'ufficio, o al periodo ecc.).

Reporting sullo stato di funzionalità/test

E' questo il reporting strettamente relativo alla gestione delle check-list statiche.
I report principali in quest'area sono :

- Elenco funzionalità esistenti
  E' una lista di tutte le funzionalità esistenti, raggruppate o eventualmente selezionate in base ad un carattere qualificatore. Ad ogni funzionalità può essere associato (opzionalmente) lo elenco di tutti i test che la coprono, raggruppati per complex (o altro classificatore) di appartenenza.
  In fig.1 è fornito un esempio di tale report.

- Elenco di tutti i complex esistenti
  E' una lista analoga a quella precedente: ad ogni complex (o altro elemento classificatore) può opzionalmente essere associato l'elenco di tutti i test che gli appartengono, e per ciascuno di questi possono essere indicate tutte le funzionalità coperte.
  Un esempio è visibile in fig. 2.

- Elenco dei test esistenti
  Si tratta ancora della lista di tutti i test esistenti, limitata eventualmente ai soli test di un complex, con indicazione di eventuali parametri caratteristici del test (es.durata).
  Per ciascun test può essere, opzionalmente, fornita la lista di tutte le funzionalità coperte.
  Per un esempio si veda la fig. 3.

- Elenco di tutte le funzionalità non coperte
  Si tratta dell'elenco di tutte le funzionalità per le quali non esiste alcun test che le copra.

- Elenco di tutti i complex privi di almeno un test

FUNZIONALITA'		TESTS	
CODICE	DESCRIZIONE	COMPLEX	NOME-TEST
F04030	CRU096 96 COL. ;	XBPRIF	TAPEXXXTAPE
			INDEXXXINDEX
		XTDGX2	DEVINDXDEVIND2
			DEVINDXDEVIND6
			DEVINDXDEVIND8
			DEVINDXDEVIND9
			DEVINDXDEVIND12
			DEVINDXDEVIND7
F04040	MFU096 MULTIFUNCTION;		
F04041	MFU096 AS CARD READER;	XBPRIF	TAPEXXXTAPE
			INDEXXXINDEX
		XTDGX2	DEVINDXDEVIND2
			DEVINDXDEVIND6
			DEVINDXDEVIND8
			DEVINDXDEVIND9
			DEVINDXDEVIND12

.Fig. 1

COMPLEX		TESTS	FUNZIONALITA'	
CODICE	DESCRIZIONE		CODICE	DESCRIZIONE
XBPRIF	PRFILE UTILITY;	TAPEXXXTAPE	F00100	CPU (GENERIC MAIN MEMORY);
			F01000	DISK (GENERIC DEVICE)MONOVOLUME;
			F02001	PRU001 (H112);
			F02002	PRU002 (TN340);
			F03001	CLUSTER 9 TRACKS;

.Fig. 2

TESTS	PARAMETRI	FUNZIONALITA'	
		CODICE	DESCRIZIONE
COBOLEMIOINXEMIO	0030	F00100	CPU (GENERIC MAIN MEMORY);
		F01000	DISK (GENERIC DEVICE)MONOVOLUME;
		F05001	CONSOLE ;
COBOLMULTIXXMULTI	0100	F00100	CPU (GENERIC MAIN MEMORY);
		F01001	DISK (GENERIC DEVICE)MULTIVOLUME;
		F05001	CONSOLE ;
		F20010	MULTIPROGRAMMING ;

Fig. 3

Gestione delle check-list dinamiche

Con il termine di "check-list dinamica" intendiamo il complesso di relazioni esistenti tra le funzionalità e i test che le coprono, da un lato, e i piani di esecuzione dei test ed i risultati effettivi di tali test, dall'altro.
Essa è caratterizzata soprattutto da relazioni del tipo :

- il test A deve essere eseguito nell'ambiente di configurazione G
- il test A è stato/non è stato eseguito nell'ambiente della configurazione G, con esito positivo/negativo/indefinito
- la funzionalità Z è stata coperta dal test B nella configurazione H, con esito positivo/negativo/indefinito
- la funzionalità X non è stata coperta da nessun test che abbia avuto esito positivo
  ecc.

La check-list è in questo caso "dinamica" in quanto fornisce una immagine di una situazione in evoluzione. Essa è quindi idonea ad essere impiegata, in coincidenza con ciascun ciclo di testing, per:

- pianificare gli ambienti (configurazioni) in cui dovranno essere eseguiti i singoli test prescelti)
- tenere traccia del procedere delle esecuzioni, in relazione a quanto pianificato inizialmente
- determinare quali test devono essere ripetuti e in quale ambiente, in considerazione del risultato ottenuto
- tradurre i dati sull'andamento delle esecuzioni dei test e sui loro risultati in termini di copertura effettiva realizzata sulle varie funzionalità.

Per la sua natura, la check-list dinamica è soggetta ad operazioni di aggiornamento del tipo:

- registrazione, cancellazione, variazione delle configurazioni previste per l'esecuzione di ciascun test
- registrazione e cancellazione dei risultati delle esecuzioni dei vari test, comprese le informazioni sull'ambiente (configurazione a livello di System Disk).

Il problema principale nell'esecuzione di queste registrazioni, consiste nell'innestare tutte le nuove informazioni sulla stessa rete di legami che caratterizza la check-list statica. TMSS provvede a tutto questo cosicchè, al termine di ogni singolo aggiornamento, i nuovi dati risultano già inseriti ed accessibili attraverso uno qualunque dei legami definiti. In particolare:

- registrazione-cancellazione delle configurazioni
  Il significato di questa operazione è quello di memorizzare un piano dettagliato di esecuzioni per ogni singolo test.
  TMSS provvede ad associare ad ogni test l'elenco delle configurazioni previste per il suo lancio; in virtù dei legami già esistenti tra il test e le funzionalità da esso coperte, queste ultime possono così a loro volta essere messe in relazione con le configurazioni previste per la loro prova.
- registrazione dei risultati
  In questo caso TMSS provvede ad associare ai singoli test i risultati delle loro esecuzioni. I risultati vengono registrati separatamente per le varie configurazioni, ed appaiono raggruppati per livello di System Disk.
  Per ciascuna configurazione in ciascun livello di S.D. compare solo l'esito dell'esecuzione più recente.
  Anche in questo caso i dati sulle esecuzioni dei test vengono direttamente inseriti e sono quindi accessibili sulla rete di legami che conducono fino al complex di appartenenza ed alle funzionalità coperte.
  E' da sottolineare il fatto che TMSS consente di prelevare automaticamente i risultati dei test dai report files generati da TRP (Test Reporting Package) [1].

Reporting sull'avanzamento del testing

Si tratta del report direttamente collegato alla gestione della check-list dinamica.

Lo scopo principale dei report di questa area è quello di fornire un'immagine aggiornata e dettagliata dell'andamento di una fase di testing, così come essa evolve. L'immagine di tale andamento è fornita sia in relazione allo stato delle esecuzioni dei test, con i loro esiti, sia in relazione allo stato di copertura effettiva delle funzionalità, così come questo si determina in base alle esecuzioni dei test.
I report principali sono :

- Stato delle esecuzioni dei test
  Il report esprime lo stato di avanzamento delle esecuzioni di tutti o di specifici test, col raffronto tra quanto previsto e quanto attuato.
  Il report ha la forma di una serie di matrici, ciascuna relativa a un test, in cui vengono evidenziate su un lato le configurazioni previste, e sull'altro i livelli di System Disk usati, con un cross-reference reciproco basato sulle esecuzioni dei test. Il report indica esplicitamente i casi in cui un test risulta "NON PREVISTO" per l'esecuzione in nessuna configurazione, ed i casi in cui un certo livello di System Disk risulta "NON UTILIZZATO" per un certo test specifico.
  Un esempio semplificato di report di questo tipo è visibile in fig. 4.

- Stato di coperture delle funzionalità
  Il report può essere considerato il corrispondente del report precedente, applicato però al caso di tutte le funzionalità (o soltanto di alcune) presenti negli archivi del TMSS. Il report ha ancora la forma di una serie di matrici, con la differenza che in questo caso ogni matrice si riferisce ad una singola funzionalità (anzichè ad un test). Ogni matrice riporta poi informazioni della stessa natura di quelle del caso precedente, cosicchè è possibile conoscere, funzione per funzione, lo stato effettivo della sua qualificazione.

- Elenco funzionalità scoperte
  Il report costituisce una sintesi del report precedente, orientato ad indicare tutte e sole le funzionalità che non sono state oggetto

```
N O M E   T E S T :    TDGX2DEVINDXDEVIND16

S.D.   E L E N C O   C O N F I G U R A Z I O N I   P R E V I S T E   P E R   I L   T E S T
       0015 0016 0018 0019 0020 0101 0105 0120 0132 0202 0209 0253 0301
H2/01  Y    Y    N    Y    ?    Y    N         N    N    N    Y
H2/02  Y    Y    Y    N    N    Y    N    Y    Y    N    N    N    N
H3/01  N    Y    Y    Y    Y         N    Y    Y    Y    ?    ?    Y
H4/01  Y    Y    Y    Y    Y    Y    Y    Y    Y    Y    Y    Y    Y
H5/10  Y    Y    Y    Y    Y    Y    Y    Y    ?    ?    ?    Y    Y
H6/13            N O N   U T I L I Z Z A T O
```

Fig. 4

di testing con esito positivo.
Il report è diviso per livelli di System-disk, cosicchè viene segnalato il fatto che una certa funzionalità sia stata provata su un livello di System-Disk, e magari non più provata sui livelli più avanzati.
Un esempio semplificato di report di questo tipo è fornito in fig. 5.
Si noti come il report, segnalando "17/92" come primo livello di System-Disk per cui esistono funzionali scoperte, consenta di comprendere che "su tutti i S.D. precedenti tutte le funzionalità sono state provate almeno una volta, mentre quasi nessuna funzionalità è stata ancora provata sull'17/92".

7. TMSS - STRUTTURA

Lo strumento completo, nella sua versione attuale, si compone di (vedi fig.6) :

- tre files indexed, con record complementari
- un file indexed per indici secondari, relativo ad uno dei tre files precedenti
- un programma di gestione fisica e logica degli archivi, capace di operare sui detti in base ad un set adeguato di co-

```
S.D. LEVEL = 17/92        FUNZIONALITA' SCOPERTE

                          F00120  CPU (MODEL 62/20);

                          F00140  CPU (MODEL 62/40);

                          F00150  CPU (MODEL 62/50);

                          F00160  CPU (MODEL 62/60);

                          F01002  NDM11X (CONVERSION);

                          F01005  MSU305;

                          F01006  MSU306;

                          F01010  MSU310;

                          F01012  MSU330;

                          F01030  SMD (DRIVER 80MB);

                          F01060  MSU360;

                          F02003  PRUC03 (PR73);
```

Fig. 5

mandi, messi a disposizione dell'utilizzatore (vedi oltre)

- un programma di interrogazione degli archivi e di stampa dei report, anche esso pilotato dai comandi messi a disposizione dell'utente
- un macro JCL per richiamare in esecuzione l'uno o l'altro dei due programmi appena detti, con l'eventuale creazione iniziale degli archivi (la cui preallocazione è così nascosta all'utente).

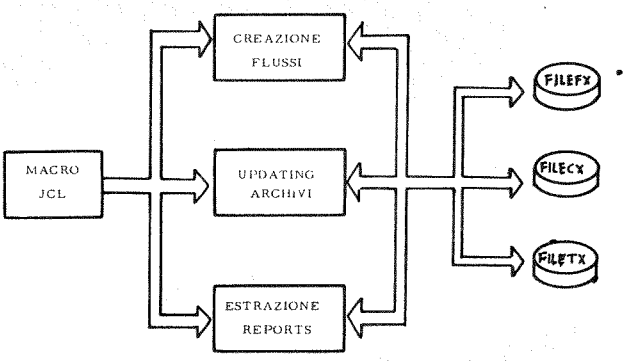


Fig. 6

Struttura degli archivi

Il TMSS dispone di tre archivi distinti :

FILEFX: contiene l'elenco di tutti gli elementi (funzionalità, ecc.)che devono essere coperti dal testing. Per ciascun elemento contiene una lista di tutti i test (divisi per complex) che lo coprono.

FILECX: contiene l'elenco di tutti i complex, con la lista dei test appartenenti a ciascuno di essi.

FILETX: contiene l'elenco di tutti i test, divisi per complex di appartenenza.
Ad ogni test è associata la lista di tutti gli elementi (tra quelli registrati in FILEFX) che ne vengono coperti.
Inoltre, ad ogni test possono essere associati :

- la lista di tutti gli ambienti sui quali se ne pianifica il lancio
- l'elenco di tutti i lanci effettuati, con indicazione del S.D. e dell'ambiente usati, oltrechè dell'esito.

Ciclo di elaborazione

Entrambi i programmi costituenti il TMSS (quello di Updating, e quello di Reporting) basano il loro funzionamento sull'elaborazione ed esecuzione di una sequenza qualsiasi di comandi, che vengono accettati uno alla volta, finchè non venga incontrata la fine dell'input.

Ogni comando è caratterizzato da una apposita stringa di caratteri (ad es.: " DEF-FX") a cui può seguire una quantità indefinita di "dati in input" per l'esecuzione del comando (nell'esempio, al comando " DEF-FX" può seguire un numero indefinito di schede che definiscono funzionalità da inserire in archivio).

8. STATO ATTUALE E POSSIBILITA' DI EVOLUZIONE

TMSS si trova attualmente in una fase iniziale del suo sviluppo, e risulta già integrato con un altro strumento disponibile a Pregnana, il Test Reporting Package - TRP, al quale si appoggia per la rilevazione automatica dei risultati dei test.

Ulteriori sviluppi del TMSS lo porteranno a trattare automaticamente check-list a più livelli, con sintesi automatica dei dati verso i livelli alti, e ne consentiranno il collegamento con un package per la gestione automatica degli archivi di test, già in corso di sperimentazione a Pregnana.

9. RIFERIMENTI

[1] Farnedi, Poggiali, Polillo, Vignoni, "TRP: un sistema per la gestione dei risultati delle prove di un sistema operativo", Congresso A.I.C.A., Milano, 1976

AVVENIMENTI

IL CONGRESSO ANNUALE DELL'AICA

Si é svolto a Pisa, nei giorni 12-14 ottobre, il Congresso annuale dell'AICA (Associazione Italiana per il Calcolo Automatico).

La manifestazione ha avuto un notevolissimo successo di pubblico grazie ad un concorso di interessanti scelte. Tra queste la principale é senza dubbio l'impegnativo e composito programma, impostato su tre sessioni parallele su argomenti di interesse ed attualità. Un secondo elemento di stimolo é stata poi la duplice mostra di prodotti industriali e dell'editoria informatica.

Parlando brevemente del programma, é stata particolarmente apprezzata la struttura che vedeva, per ciascuno dei tre giorni, una prima sessione plenaria dedicata ad uno dei tre temi principali, affidata a conferenzieri invitati, di chiara fama: T.E. Cheatam della Harvard University (Software Methodologies), N.A. Abramson della University of Hawaii e E. Douglas Jensen della Honeywell Systems and Research Center (Distributed Computer Systems), M.W. Blasgen della IBM e E.J. Neuhold della Università di Stoccarda (Data Base Systems).

Un secondo punto di interesse é stata la sessione industriale che si é affiancata a quella dedicata ai tre temi del convegno, rispondendo certamente alla curiosità di molti intervenuti circa i prodotti di più recente annuncio.

Vi sono stati infine due dibattiti di attualità: uno sull'inserimento dei laureati in informatica nel mondo del lavoro e l'altro sul tema "Economia, Società ed Informatica".

La mostra dedicata ai prodotti industriali e progetti sperimentali vedeva esposte diverse apparecchiature rivolte principalmente a impieghi di informatica distribuita. Completava in tal modo il programma delle conferenze della sessione industriale.

Per quanto riguarda la sessione che ci interessa in modo particolare, Metodologie di Software, riportiamo l'elenco degli interventi principali.

New Directions in Software Development Tools, T.E. Cheatham.

Design and Programming Methodologies and Development of Supporting Tools: Dopo la programmazione strutturata, G. Degli Antoni, B. Zonta; La programmazione strutturata: alcune riflessioni per la definizione di una disciplina scientifica, G. De Michelis, C. Simone; Un meccanismo interpretativo per il trattamento delle condizioni eccezionali in un programma gerarchico, V. De Antonellis, M. Maiocchi, R. Polillo; L'uso di coroutines in applicazioni EDP: un insieme di macro per la gestione di coroutines e semicoroutines rientranti in COBOL, O. Fouillouze, F. Giordano, R.Polillo, G. Zafferri.

Semantics of Programming Languages and Program Transformations: Il trattamento delle variabili nei sistemi di programmazione, L. Aiello, G. Prini; Sull'implementazione dell'attivazione di funzioni nei linguaggi funzionali, D. Buggiani; Una caratterizzazione della semantica dei linguaggi programmativi basata sulla valutazione simbolica, A. Pegna.

Portability of Programs: la necessità di un linguaggio intermedio come strumento universale di sviluppo del software, G. Casadei, G. Misuri, A. Teolis; Realizzazione di un bootstrap per il sistema XPL per scrivere compilatori indipendente dalla macchina, G. Dodero, L. Durante.

Testing and Validation of Programs: Structured testing: a comprehensive methodology for software verification, D.A. Walsh, La valutazione dei test ai fini della produttività nella qualificazione del software: uno strumento ed una esperienza pilota, A. Cazziol, C. Cucciati; Progettazione di un traduttore ATLAS, G. Terrevoli.

Program Documentation: Macroschemi: una proposta per la documentazione di certe classi di programmi che svolgono traduzioni o dialoghi, S. Crespi Reghizzi; GALOIS: un modo di documentare sistemi software, G.A. Lanzarone, D. Scrignaro.

Specification Languages: Module interface specification and interconnection, G. Ghezzi, P. Paolini, F. Tisato; Un linguaggio di specifica eseguibile, B. Balatresi.

Languages for Structured and Modular Programming: Alcune considerazioni ed esperienze sui linguaggi di sistema, M. Refice, M. Capurso; Tipi di dati e strutture di programmi nel SIMULA 67, S. Brandi, T. Pedrotti; LIN 3: a family of System Implementation Languages, S. Copelli, A. Osnaghi, F. Tisato.

Linguistic Constructs: Improved I/O facilities in high-level languages, S. Crespi Reghizzi, P. Della Vigna; Dynamic resource management in a language for real time programming, P. Ancilotti, M. Boari, N. Lijtmaer; An object oriented notation for path expressions, P.R. Torrigiani, P.E. Lauer.

Abstract Data Types: Un metodo per la definizione di tipi di dati astratti nel linguaggio PL/I, M. Di Manzo, A.L. Frisiani, G. Olimpo; Passing parameter types in programming languages with data abstractions, P. Asirelli, F. Gimona, A. Martelli, U. Montanari.

Si ricorda che gli atti completi del Congresso sono reperibili presso la segreteria dell'AICA, presso la FAST.

(W. Anselmetti)

NOTIZIE

Il Centro Televisivo dell'Università degli Studi di Milano (CTU) riprenderà con il nuovo anno accademico a trasmettere via etere corsi di aggiornamento post-laurea, corsi per studenti, e corsi di formazione permanente. I programmi del CTU, iniziati sperimentalmente in luglio con corsi di informatica, saranno ampliati, con il nuovo ciclo di trasmissioni, ad altre discipline.

Le trasmissioni, messe in onda grazie ad un accordo fra l'Università di Milano e la Società dei Servizi Televisivi Royal SPA, vengono ricevute per il momento in cinque regioni italiane. La Lombardia viene servita da Tele Nord Milano (canale UHF 63, orientare l'antenna su Piazza della Repubblica, Milano oppure canale UHF 58, orientare l'antenna su Monte Canate a Sud di Piacenza). Il Veneto viene servito dalla Tele Monte Faeto (canale UHF 45, orientare l'antenna su Monte Faeto, a Sud di Sassuolo). Infine la Toscana e l'Umbria vengono servite dalla T.R.E. (canale UHF 40, orientare l'antenna su Monte Secchieta a Sud-Est di Firenze; canale UHF 51, orientare l'antenna su Monte Meto in Versilia o su Monte Cetona a ovest di Chiusi; canale UHF 59, orientare l'antenna su Monte Pizzorne a nord di Lucca).

Le trasmissioni inizieranno lunedì 28 novembre 1977 per la Lombardia, e lunedì 5 dicembre per il Veneto, l'Emilia, la Toscana e l'Umbria. Quelle dedicate all'informatica copriranno, per i primi mesi, le seguenti fasce orarie: lunedì, mercoledì, e venerdì dalle 9,30 alle 10; martedì e giovedì dalle 10 alle 11 e dalle 18 alle 18,30.

Sono per ora previsti i seguenti tre corsi di informatica, che verranno ripetuti più volte:

- "Introduzione alla multiprogrammazione strutturata", a cura di R. Polillo e O. Fouillouze (circa 8 ore complessive).
- Corso di COBOL, a cura della Honeywell (circa 3,5 ore complessive)
- "Tecniche di testing dei programmi", a cura di M. Maiocchi (circa 8 ore complessive).

IL SEGNALIBRO

In questa rubrica vengono segnalati libri, articoli o studi di recente pubblicazione che, a giudizio della redazione o dei lettori di NOTE DI SOFTWARE, rivestono un particolare interesse nell'ambito dei temi a cui questa rivista é dedicata.

Le citazioni fra virgolette, se non ne é indicata la fonte, sono tratte dai lavori stessi.

Programmazione COBOL - Libri

● Chmura,L.J., Ledgard,H.F., COBOL WITH STYLE - PROGRAMMING PROVERBS (Principles of Good Programming with Numerous Examples to Improve Programming Style and Proficiency), Hayden Computer Programming Series, Hayden Book Company Inc., Rochelle Park, New Jersey, (1976), pagg. 148.

Un volumetto divertente ed estremamente utile, che meriterebbe di essere letto e discusso da chiunque programmi in Cobol. Il secondo capitolo presenta 19 proverbi, ispirati a quelli proposti in due precedenti volumetti di Ledgard ("PROGRAMMING PROVERBS, with sample programs in PL/I, Algol, Fortran and BASIC", e "PROGRAMMING PROVERBS FOR FORTRAN PROGRAMMERS" stessa collana), e qui illustrati con semplici ed efficaci esempi in Cobol.
I proverbi sono: 1. Don't Panic, 2. Define the Problem Completely, 3. Start the Documentation Early, 4. Think First, Code Later, 5. Proceed Top-Down, 6. Beware of Other Approaches, 7. Code in Logical Units, 8. Use PERFORM and CALL, 9. Don't GO TO, 10. Prettyprint, 11. Comment Effectively, 12. Use Mnemonic Words, 13. Get the Syntax Correct Now, 14. Plan for Change, 15. Don't Leave the Reader in the Dust, 16. Prepare to Prove the Pudding, 17. Have Someone Else Read the Work, 18. Read the Manual Again, 19. Don't Be Afraid to Start Over.
Il capitolo 3 discute la programmazione top-down attraverso un unico divertente esempio; il capitolo 4 fornisce uno standard di codifica Cobol costituito da una ventina di regole; il capitolo 5 ("Odds and Ends") conclude il manualetto con la discussione di alcuni temi specifici: come scegliere identificatori mnemonici, il problema della efficienza, l'uso dei flowcharts, ecc. (R.P.)

● Weinberg,G.M., Wright,S.E., Kauffman,R., Goetz,M.A., HIGH LEVEL COBOL PROGRAMMING, Winthrop Computer Systems Series, Winthrop Publishers Inc., Cambridge (Massachusetts), 1977, pagg. XX+252

"...This book is for <u>all</u> people involved in the development of COBOL programs - from top management on down.[...] In a way, the entire book is about the <u>interweaving of technical and business questions</u>. Managers must be willing to bear with us through some demonstrations of why their best-laid plans can be sabotaged by poor attention to technical details. On the other hand, programmer/analyst types are going to have to learn more about how their work fits into a larger world than MOVE CORRESPONDING or GO TO DEPENDING ON..."
Indice: A. The Programming Business; B. Managing High Level Programming; C. Critical Program Reading; D. High Level Control; E. Style and Workmanship; F. Modularity; G. Refinement; H. Verification; I. Sheltering; J. Maintenance; K. Putting Theory into Practice; Appendix 1.: The MetaCOBOL System; Appendix 2.: The LIBRARIAN System.

La nuova serie di Informatica della collana Testi Scientifici Molulari ISEDI

"Questa serie, diretta da G. Degli Antoni, D. Ferrari, F.P. Preparata, M. Sce, si propone di offrire, attraverso una sequenza di moduli coordinati, un quadro sufficientemente completo dei fondamentali argomenti dell'ingegneria dei calcolatori e dell'Informatica teorica. I testi si propongono come strumento per le Facoltà universitarie e per corsi di formazione aziendale, per l'aggiornamento e lo studio da parte degli operatori del settore, per corsi destinati alla formazione di Docenti di scuola superiore. La trattazione si articola attraverso un discorso iniziale preciso ma di ampio respiro che presenta gli aspetti intuitivi della materia per poi svilupparsi attraverso rigorose definizioni e dimostrazioni".
Sono finora usciti i primi due volumi:

● Bovet,D.P., SISTEMI OPERATIVI, collana TSM, Serie di Informatica, 1, ISEDI, Milano (1977), pagg. 279, Lit. 10.000

"Il libro costituisce una introduzione avanzata allo studio dei sistemi operativi i quali, adattandosi all'hardware esistente, hanno lo scopo di ottimizzarne le prestazioni. Nel testo vengono introdotte gradualmente le principali tecniche usate nella realizzazione di detti sistemi. Sommario: 1. Caratteristiche funzionali - 2. Interazione tra processi - 3. Supervisore - 4. Gestione di una singola risorsa - 5. Gestione di un sistema di risorse - 6. Sistemi di archiviazione - 7. Sistemi di protezione 8. Comunicazione tra l'utente e il sistema operativo - 9. Ordinamento dei lavori - Appendice - Bibliografia." (copert.)

● Galimberti,R., Rosci,G., SOFTWARE DI BASE E ASSEMBLATORI, collana TSM, Serie di Informatica, 2, ISEDI, Milano (1977), pagg. 223, Lit. 9.000

"Nel libro sono sviluppati vari temi a partire dalla struttura degli elaboratori elettronici, attraverso l'illustrazione delle tecniche di programmazione strutturata, sino allo studio e realizzazione del Software di Base, con particolare riferimento ad assemblatori, linkers, loaders, programmi di debugging ed editors. Sommario: 1. Elaboratori, algoritmi, programmi - 2. Sistemi di numerazione, codici, rappresentazione dei dati - 3. Struttura dell'elaboratore - 4. Tecniche di implementazione - 5. Il sistema PDP 11 - 6. Il sistema IBM 370 - 7. Modalità di indirizzamento e sottoprogrammi - 8. Programmi per programmare - 9. Linguaggio di Assemblatore - 10. Programma Assemblatore - 11. Programmi di caricamento - 12. Il sistema di ingresso -uscita - Bibliografia." (copertina)

Libri sulla programmazione strutturata

● Lanzarone,G.A., Maiocchi,M., Polillo,R., INTRODUZIONE ALLA PROGRAMMAZIONE STRUTTURATA, Franco Angeli Editore (Collana dei Quaderni di Informatica), 1977, pagg. 221, Lit. 6.500.

"Una guida concettuale e operativa alla programmazione strutturata, diretta tanto a chi si accosta per la prima volta alla programmazione quanto a chi ha maturato un'esperienza tradizionale in questa attività.
Nei primi capitoli, dopo una breve introduzione ai concetti fondamentali della programmazione, vengono definiti gli strumenti linguistici atti a rappresentare algoritmi in modo ordinato. In particolare, viene presentato un insieme flessibile di costrutti di controllo per strutturare i programmi, indicandone per ciascuno e comparativamente il campo di applicabilità e illustrando come essi si possano realizzare in alcuni dei più diffusi linguaggi procedurali correnti, quali il Fortran, il Cobol e un Assembler tipico.
Nei capitoli successivi vengono affrontati gli aspetti relativi all'introduzione sistematica di commenti nel testo di un programma, che esprimano il significato funzionale di una porzione di codice o lo stato della computazione in particolari punti del programma. Viene fornito, a partire dalle strutture di controllo introdotte, uno standard di documentazione preciso seppure non vincolante per il programmatore. Si mostra poi come tali asserzioni possono essere usate a base di una verifica, intuitiva, della congruenza logica delle varie parti di programma, e si specificano e illustrano delle regole informali, che possono essere seguite per la revisione della consistenza di un programma scritto con i costrutti di controllo introdotti.
Negli ultimi capitoli viene trattato il problema della costruzione di programmi più

complessi, secondo il metodo della specificazione per successivi livelli di dettaglio. Viene mostrato come impiegare gli strumenti linguistici prima considerati per attuare un procedimento che, a partire da un problema dato, conduca gradualmente e in modo controllato, attraverso successive approssimazioni, fino a un testo strutturato e documentato che costituisca il programma desiderato." (copertina)

Strumenti per la prova dei programmi

● Howden, W.E. SIMBOLIC TESTING AND THE DISSECT SYMBOLIC EVALUATION SYSTEM, IEEE Transactions on Software Engineering SE-3, 4 (luglio 1977), 266-278.

"Symbolic testing and a symbolic evaluation system called DISSECT are described. The principal features of DISSECT are outlined. The results of two classes of experiments in the use of symbolic evaluation are summarized. Several classes of program errors are defined and the reliability of symbolic testing in finding bugs is related to the classes of errors. The relationship of symbolic evaluation systems like DISSECT to classes of program errors and to other kinds of program testing and program analysis tools is also discussed. Desirable improvements in DISSECT, whose importance was revealed by the experiments, are mentioned."
(DISSECT permette l'esecuzione simbolica di programmi FORTRAN ANSI). Un lavoro assai interessante, soprattutto per l'analisi degli esperimenti tendenti a mettere in luce la differenza dei risultati ottenibili con tecniche di valutazione simbolica e tecniche di prova usuali (esercizio del programma con dati campione).
"...In summary, symbolic testing was found to be somewhat more reliable than actual data testing for computational errors that affect a program's path functions. In the sample of 12 incorrect programs, a testing strategy which used actual data would have been reliable or probably reliable for discovering 11 of 22 errors and symbolic testing for 15 of the 22 errors. Symbolic testing was not reliable for path-domain errors or for certain kinds of data flow anomalies..."

● Hamlet, R.G. TESTING PROGRAMS WITH THE AID OF A COMPILER, IEEE Transactions on Software Engineering SE-3, 4 (luglio 1977), 279-290

Presenta uno strumento per il testing dei programmi che impone di specificare, nel codice stesso delle procedure che devono essere provate, l'insieme dei test da effettuare.
Tali specifiche sono introdotte mediante frasi (interpretate come commenti quando il programma viene compilato per esecuzione normale) della forma:

$$/+\text{TEST}(x_1,y_1)\ldots(x_n,y_n)+/.$$

Ogni coppia di valori (x_i,y_i) specifica che il componente in esame deve venire eseguito con il valore di ingresso x_i, nel qual caso il valore di uscita atteso é y_i.
(Il lavoro descrive una realizzazione prototipale in cui si suppone che i componenti siano funzioni intere, con un solo argomento intero).
"Programs which carry adequate tests with them in this way should be resistant to maintenance errors. If the specifications are independent of program detail they are easy to give, and unlikely to contain errors in common with the program".
Il compilatore attrezza il programma sorgente in modo opportuno, e lo collega ad un driver che controlla l'esecuzione delle prove, richiamando ogni componente con i valori di input specificati nella sua clausola TEST, nell'ordine specificato, e fornendo, in modo interattivo, l'esito di ogni singola prova. Vengono fornite inoltre informazioni sulla "completezza" dei tests specificati dal programmatore (statements non eseguiti, ecc.). Se, durante la prova di un componente X, questo richiama un componente Y, con il valore di chiamata v, Y non viene eseguito: ad X viene trasmesso il valore di output corrispondente, nella clausola TEST di Y, al valore di v (se la coppia di valori é stata specificata). (R.P.)

Strumenti per il debugging e l'analisi dei programmi

● Cohen,J., Carpenter,N., A LANGUAGE FOR INQUIRING ABOUT THE RUN-TIME BEHAVIOUR OF PROGRAMS, Software-Practice and Experience 7, 4(Luglio-agosto 1977), 445-460

"This paper describes a language for studying the behaviour of programs, based upon the data collected while these programs are executed by a computer. Besides being a useful tool in debugging, the language is also valuable in the experimental evaluation of the complexity of algorithms, in studying the interdependence

of conditionals in a program and in determining the feasibility of transporting programs from one machine to another. The program one wishes to analyse is written in an Algol 60-like language; when the program is executed it automatically stores, in a data base, the information needed to answer general questions about computational events which occurred during execution. This information consists (basically) of the list of labels passed while the program is being executed, and the current values of the variables. Since the list of labels is describable by regular expressions, these expressions can also be used to identify specific subparts of the list and therefore allow access to the values of the variables. This constitutes the basis for the design of the inquiry language. The user's questions are automatically answered by a processor which inspects the previously generated data base. The paper also presents examples of the use of the language and describes the implementation of its processor."

Fortran

● Woolley,J.D., FORTRAN: A COMPARISON OF THE NEW PROPOSED LANGUAGE (1976) TO THE OLD STANDARD (1966), SIGPLAN Notices (ACM) 12, 7 (luglio 1977), 112-125

"This document is intended to rapidly acquaint the reader with the changes that have occurred in the Fortran language in the past decade. It assumes some familiarity with the Fortran currently in use."

Metodologie di programmazione - articoli vari

● Coleman,D., THE SYSTEMATIC DESIGN OF FILE-PROCESSING PROGRAMS, Software-Practice and Experience 7, 3 (giugno-luglio 1977), 371-381.

"The theory of formal languages is applied to the systematic top-down design of file processing programs. A methodology is described which leads to well structured programs for a large class of data processing problems. The method leads to more reliable programs because it is easier to check the method steps than an intuitive program. The method and its limitations are illustrated by the design process for some data processing programs. The limitations of the method are explained as the result of a theorem about formal languages." Il metodo permette di modellare un programma sulla struttura dei suoi dati di ingresso.
"The stages proposed for the systematic design of file-processing programs are:

1. specify a set of 'BNF' productions describing the structure of the input file;
2. construct a parser program for the input file using the program construction rules;
3. relate the processing of input records and the generation of output records to the syntactic structure of the input file;
4. insert the processing and output statements into the parser;
5. code the resulting program in the programming language to be used."

"...Program design based on the syntactic structure of files is not new, for it has been used in compiler writing for many years, as well as having been the basis of much of our intuition. And, of course, it also bears a close relationship to the basic design method of Jackson. What is different is that this presentation makes available the results from the rigorous area of formal language theory to the practice of program design. As a consequence, the results are checkable and limitations of the method are quite explicable and do not appear somewhat mystical and arbitrary."

● Yeh,R.T. (editor), CURRENT TRENDS IN PROGRAMMING METHODOLOGY, vol. I: SOFTWARE SPECIFICATION AND DESIGN, Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, (1977), pagg. XI+275

"The purpose of this series in programming methodology is intended to bring together a collection of tutorial papers in each volume which are representative of the current trends in one of the above menttioned specific subject areas of the programming activities [specifications, design, validation, modeling and structuring of programs and data]. The intention of this first volume is to survey recent developments of programming principles and techniques for the systematic design of well-structured and reliable software architecture".

Il volume comprende nove lavori, quasi tutti già apparsi altrove e assai noti: "An introduction to formal specifications of data abstractions", di B. Liskov e S. Zilles; "Languages and structured programs", di W.A. Wulf; "A formal methodology for the design of operating system software" di L. Robinson, K.N. Levitt, P.G. Neumann e A.R. Saxena; "The influence of Software structure on reliability" di D.L. Parnas; "On the development of large reliable programs" di R.C. Linger, H.D. Mills; "Structured programming with GO TO statements", di D.E. Knuth; "System structure for software fault tolerance", di B. Randell; "Control-Record-Driven processing", di P. Naur; "Guarded commands, nondeterminancy and formal derivation of programs", di E.W. Dijkstra.
Segue una bibliografia commentata, più di 200 titoli, compilata da A.B. Marmor-Squires.

● Ledgard,H., Singer,A., Hueras,J., A BASIS FOR EXECUTING PASCAL PROGRAMMERS, SIGPLAN Notices (ACM), vol. 12, n. 7 (luglio 1977), 101 - 105

Uno standard per la codifica di programmi Pascal.

● Shneiderman,B., Mayer,R., McKay,D., Heller,P., EXPERIMENTAL INVESTIGATIONS OF THE UTILITY OF DETAILED FLOWCHARTS IN PROGRAMMING, Communications of the ACM 20, 6 (giugno 1977), 373-381

"This paper describes previous research on flowcharts and a series of controlled experiments to test the utility of detailed flow charts as an aid to program composition, comprehension, debugging, and - modification. No statistically significant difference between flowchart and non-flowchart groups has been shown, there by calling into question the utility of detailed flowcharting. A program of further research is suggested."
Gli esperimenti descritti sono stati effettuati con studenti (corsi di programmazione, di livello introduttivo o intermedio). Il linguaggio usato é il Fortran.

Ingegneria del software - libri

● Tausworthe,R.C., STANDARDIZED DEVELOPMENT OF COMPUTER SOFTWARE, Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632 (1977), pagg. XIII + 379.

Un libro dall'indice alquanto insolito, che riunisce temi raramente compresenti in uno stesso libro: 1. Introduction, 2. Fundamental Principles and Concepts, 3. Specification of Program Behaviour, 4. Program Design, 5. Structured Non Real-Time Programs (una trattazione ampia e non formale sui flow-charts strutturati, compreso il caso delle uscite anomale), 6. Real-Time and Multiprogrammed Structured Programs (con una semplice notazione che estende i flow-charts usuali), 7. Control-Restrictive Instructions for Structured Programming (CRISP) (il caso di linguaggi non strutturati, ai quali vengono aggiunte opportune strutture di controllo mediante precompilatori o macro), 8. Decision Tables as Programming Aids, 9. Assessment of Program Correctness, 10. Project Organization and Management.

DP auditing

● Il numero di agosto di DATAMATION (vol. 23, n.8, 1977) contiene tre articoli sui metodi di "audit": Perry,W.E. e Fitz-Gerald,J., DESIGNING FOR AUDITABILITY; Wilkinson,B., AN APPLICATION AUDIT; Yasaki, E.K., WHO IS THE DP AUDITOR?

Un libro sulle tavole di decisione

● Metzner,J.R., Barnes,B.H., DECISION TABLE LANGUAGES AND SYSTEMS, ACM Monograph Series, Academic Press Inc., (1977), pagg. VIII+172.

Multiprogrammazione

● Agerwala, T., SOME EXTENDED SEMAPHORE PRIMITIVES, Acta Informatica 8, 3 (1977), 201-220

"This paper presents a proposal for synchronizing primitives obtained as an extension of Dijkstra's P, V primitives. The extended primitives are shown to be complete: they can represent any desired interaction between processes without the use of conditionals. The usefulness of these primitives is illustrated by presenting simple solutions to a series of coordination problems of increasing complexity. Two selected problems are used to illustrate disadvantages of existing synchronizing mechanisms. The extended primitives shift some of the burden from the

programmer to the system since they are easier to use but more difficult to implement. However, even though each primitive operation may take longer to execute (as compared to the simple P, V primitives) the total system overhead can be substantially less especially for complex coordination problems. The paper presents a straightforward but efficient implementation of the extended primitives."
Le primitive proposte sono:

$pe(S_1, S_2, \ldots, S_n, S_{n+1}, \ldots, S_{n+m})$
if for all i, $1 \leq i \leq n$, $S_i > 0$ **and for all** j, $1 \leq j \leq m$, $S_{n+j} = 0$
then for all i, $1 \leq i \leq n$, $S_i := S_i - 1$
else the process is blocked
$ve(S_1, S_2, \ldots, S_n)$
for all i, $1 \leq i \leq n$, $S_i := S_i + 1$.

Programmazione per tempo reale

● Wirth,N., TOWARD A DISCIPLINE OF REAL-TIME PROGRAMMING, Communications of the ACM 20, 8 (agosto 1977), 577-583.

"Programming is divided into three major categories with increasing complexity of reasoning in program validation: sequential programming, multiprogramming, and real-time programming. By adhering to a strict programming discipline and by using a suitable high-level language molded after this discipline, the complexity of reasoning about concurrency and execution time constraints may be drastically reduced. This may the the only practical way to make real-time systems analytically verifiable and ultimately reliable. A possible discipline is outlined and expressed in terms of the language Modula".

Data abstraction

● Il numero di giugno di Communications of the ACM (vol. 20, n.6, giugno 1977) propone cinque lavori selezionati fra quelli presentati alla "Conference on Data: Abstraction, Definition and Structure" (22-24 marzo 1976): Ledgard,H.F., Taylor,R.W., TWO VIEWS OF DATA ABSTRACTION; Zloof, M.M., de Jong,S.P., THE SYSTEM FOR BUSINESS AUTOMATION (SBA): PROGRAMMING LANGUAGE; Guttag,J., ABSTRACT DATA TYPES AND THE DEVELOPMENT OF DATA STRUCTURES; Smith,J.M., Smith, D.C.P., DATABASE ABSTRACTION: AGGREGATION; Gries,D., Gehani,N., SOME IDEAS ON DATA TYPES IN HIGH-LEVEL LANGUAGES.

● Liskov, B., Snyder,A., Atkinson,R., Schaffert,C.,ABSTRACTION MECHANISMS IN CLU, Communications of the ACM 20, 8 (agosto 1977), 564-576.

"CLU is a new programming language designed to support the use of abstractions in program construction. Work in programming methodology has led to the realization that three kinds of abstractions-procedural, control, and especially data abstractions-are useful in the programming process. Of these, only the procedural abstraction is supported well by conventional languages, through the procedure or subroutine. CLU provides, in addition to procedures, novel linguistic mechanisms that support the use of data and control abstractions. This paper provides an introduction to the abstraction mechanisms in CLU. By means of programming examples, the utility of the three kinds of abstractions in program construction is illustrated, and it is shown how CLU programs may be written to use and implement abstractions. The CLU library, which permits incremental program development with complete type checking performed at compile time, is also discussed".

Didattica della programmazione

● Horning, J.J., Wortman, D.B., SOFTWARE HUT: A COMPUTER PROGRAM ENGINEERING PROJECT IN THE FORM OF A GAME, IEEE Transactions on Software Engineering SE-3, 4 (luglio 1977), 325-330.

"This report describes one of the most successful course project assignments that we have ever given. Its success came neither from the content of the project (which was quite ordinary) nor from the nature of the course. Rather, it is attributable to the form of the project, which was that of a multiplayer, not-quite zero-sum, game. Since this form could easily be adapted to other courses in other environments, it seems worth describing it and discussing the factors that contributed to its success".
Le regole di Software Hut ("a small software house"), sono:
"Three persons comprise a Software Hut. Huts are to be formed by Day 1.
Phase A (DAYS 1-29)
Each hut is to design, implement, validate,

document, and otherwise prepare for sale either Module X or Module Y of the attached specification.

Phase B (DAYS 29-50)

Each hut is to write a driver program and integrate Modules X and Y purchased from two other huts, and prepare the resulting system for sale.

Phase C (DAYS 50-64)

Each hut is to purchase a system (not containing its own module) from another hut, modify it according to revised specifications (to be provided on Day 53), and prepare the revised system for sale.

Scale Of Prices

The Bank will pay the following prices for successful completion of each phase by the indicated date:

Quality	Price
A+	300 Ped
A	250 Ped
A-	200 Ped
B+	150 Ped
B	100 Ped
B-	50 Ped
C	0 Ped

(Late completion will be penalized 25 Ped/day) (Ped=Program engineering dollar).
These prices will be added to the hut's trading profit (or loss) at the end of phases A and B as indicated by sales recorded with the Bank."
Gli autori descrivono la loro esperienza con diverse varianti del gioco in corsi "graduate" all'Università di Toronto, e forniscono indicazioni per il suo miglioramento.

● Holt,R.C., Wortman,D.B., Barnard,D.T., Cordy,J.R. SP/k: A SYSTEM FOR TEACHING COMPUTER PROGRAMMING, Communications of the ACM 20, 5 (maggio 1977), 301-309

"SP/k is a compatible subset of the PL/I language that has been designed for teaching programming. The features of the SP/k language were chosen to encourage structured problem solving by computers, to make the language easy to learn and use, to eliminate confusing and redundant constructs, and to make the language easy to compile. The resulting language is suitable for introducing programming concepts used in various applications, including business data processing, scientific calculations and non-numeric computation. SP/k is actually a sequence of language subsets called SP/1, SP/2, ...,SP/8. Each subset introduces new programming language constructs while retaining all the constructs of preceding subsets. Each subset is precisely defined and can be learned or implemented without the following subsets."

Una nuova disciplina: la scienza del software

● Halstead,M.H., ELEMENTS OF SOFTWARE SCIENCE, Elsevier Computer Science Library, Elsevier North Holland Inc., (1977), pagg. XIV+127.

Il libro costituisce la prima esposizione sistematica del lavoro condotto da Halstead e, indipendentemente, da numerosi altri ricercatori, consistente nel tentativo di introdurre una nuova attitudine nello studio del software, rispetto a quella tradizionale della Computer Science. L'oggetto di studio é l'attività umana di preparazione di programmi per elaboratore; l'attitudine che l'A. si propone di adottare é quella tradizionale delle Scienze Naturali: cercare di ricavare da osservazioni sperimentali sistematiche leggi capaci di formulare previsioni sui fenomeni considerati, e capaci di costituire un campo scientifico in grado di "spiegare" l'insieme dei fenomeni che sono oggetto di studio. Si tratta di un programma ambizioso che é consistito principalmente finora nell'identificazione delle osservabili elementari dei fenomeni riguardanti la preparazione dei programmi, nella derivazione e validazione statistica di un insieme di equazioni che legano tali osservabili a una serie di parametri di significato intuitivo, come la lunghezza di un programma, il suo volume (inteso come numero minimo di bits necessari a contenerlo), il livello del linguaggio, lo sforzo di programmazione, inteso come tempo minimo necessario a un ipotetico programmatore esperto per implementare l'algoritmo, ed altri. Le osservabili elementari scelte dall'A. sono il numero totale degli operatori e degli operandi presenti in un programma, e il numero degli operatori e degli operandi distinti, ovvero il vocabolario con cui il programma é costruito. Il carattere di queste osservabili rende evidente il tipo di osser-

vazioni e misurazioni elementari eseguibili su un programma, consistenti principalmente in operazioni di conteggio. Se da una parte appaiono decisamente rilevanti la corrispondenza tra dati sperimentali e dati calcolati a partire dalle equazioni ipotizzate, dall'altra emergono numerose discrepanze, che indicano che c'é ancora molto lavoro da fare per dare una veste scientifica rigorosa alla nascente disciplina della Scienza del Software. Ci riferiamo in particolare all'ambiguità che ancora sussiste nella definizione in questo contesto del concetto di operando e di operatore nei vari linguaggi di programmazione, e alla ambiguità che sussiste nel processo di derivazione di alcune equazioni fondamentali, come quella della lunghezza e del volume.
Nonostante questi difetti, i problemi che l'autore apre sono indubbiamente stimolanti, in particolare quelli riguardanti i possibili sviluppi applicativi di questa disciplina, come la possibilità di derivare dalla sola conoscenza di alcuni parametri la lunghezza del programma prima del suo sviluppo e lo sforzo di programazzione necessario. (D. Marini)

Text processing

● Schneider,B.R.jr, Watts,R.M., SITAR: AN INTERACTIVE TEXT PROCESSING SYSTEM FOR SMALL COMPUTERS, Communications of the ACM 20, 6 (giugno 1977), 495-499

"SITAR, a low-cost interactive text handling and text analysis system for nontechnical users, is in many ways comparable to interactive bibliographical search and retrieval systems, but has several additional features. It is implemented on a PDP/11 time-sharing computer invoked by a CRT with microprogrammed editing functions. It uses a simple command language designating a function, a file, and a search template consisting of the textual string desired and strings delimiting the context in which the hit is to be delivered. Extensive experience with SITAR shows that the combined powers of simple structure, string orientation, circular file structure, a CRT with local memory, and conversational computing produce a system much more powerful than the sum of its parts."

Information retrieval - un libro

● Paice, C.D., INFORMATION RETRIEVAL AND THE COMPUTER, Series: Computer Monographs, Vol.26 MacDonald and Jane's Publishers Ltd., London (1977), pgg. X+206, tables.

Nella fortunata serie MacDonald che dal 1967 propone agili monografie di Informatica, esce ora questo volume di C.D. Paice sull'Information Retrieval. L'Information Retrieval (il reperimento delle informazioni) è diventata una branca autonoma dell'Informatica, in conseguenza del fatto che le informazioni sono in tale quantità e crescono con tale velocità, in un qualunque settore di attività, da rendere complesso e costoso anche il semplice compito di organizzarle in modo che poi siano reperibili, e da richiedere di conseguenza l'ausilio del calcolatore.
Sono state proposte negli ultimi anni tecniche diverse per la classificazione, l'ordinamento ed il ritrovamento delle informazioni di un archivio di grandi dimensioni.
Il libro di Paice presenta in modo chiaro e comprensibile un repertorio vasto ed aggiornato di queste tecniche. Dopo aver introdotto le nozioni generali su files, records e codifica delle informazioni egli descrive le tecniche di classificazione e indiciamento dei documenti. Le prime servono per organizzare l'immagazzinamento delle informazioni all'interno di un archivio, gli indici di contro forniscono le chiavi per il reperimento dell'informazione in un archivio. Nelle ultime tre parti infine egli affronta il problema dell'automazione di queste operazioni. Vengono quindi descritte tecniche di classificazione automatica (basate sull'uso di parole chiave, sull'analisi della frequenza delle parole nei documenti ecc.) tecniche di estrazione automatica di riassunti, metodi di gestione automatica di un archivio.
Ogni tecnica proposta é descritta in modo chiaro e non é necessario essere esperti programmatori per capire il libro.
In conclusione un pregevole repertorio di tecniche, utile a chi voglia avere una panoramica sull'argomento. Manca invece un'impostazione più generale del problema che inserisca l'Information Retrieval di archivi di documenti, nel più generale caso dell'Information Retrieval di insiemi di informazioni, relazionate tra di loro.
(G. De Michelis)

Honeywell

Honeywell Information Systems Italia

Via Laboratori Olivetti - Pregnana Milanese

UNIVERSITA'
DEGLI STUDI DI MILANO

ISTITUTO DI CIBERNETICA

Via Viotti, 5 - Milano